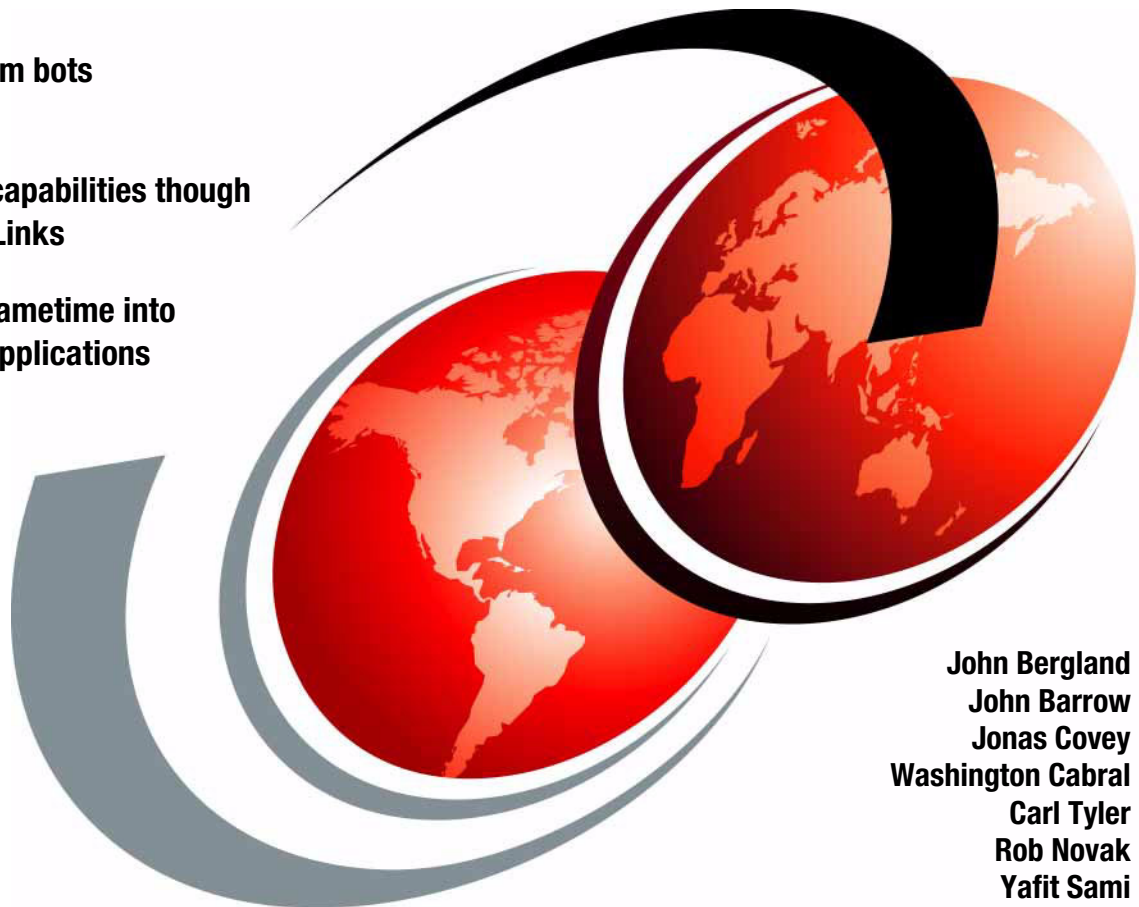


Lotus Instant Messaging/ Web Conferencing (Sametime): Building Sametime-Enabled Applications

Build custom bots

**Advanced capabilities through
Sametime Links**

**Integrate Sametime into
workflow applications**



**John Bergland
John Barrow
Jonas Covey
Washington Cabral
Carl Tyler
Rob Novak
Yafit Sami**



International Technical Support Organization

**Lotus Instant Messaging/Web Conferencing
(Sametime): Building Sametime-Enabled
Applications**

November 2003

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (November 2003)

© Copyright International Business Machines Corporation 2003. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xi
Trademarks	xii
Preface	xiii
The team that wrote this redbook	xiii
Become a published author	xvi
Comments welcome	xvi
Part 1. Introduction	1
Chapter 1. Overview of Sametime	3
1.1 Benefits and importance of collaboration	5
1.2 What is Sametime?	5
1.3 Sametime services	6
1.3.1 Real-time collaboration: community services	6
1.3.2 Sametime online meeting services	8
1.3.3 Sametime customization and integration services	9
1.4 Overview of the Sametime 3.1 Toolkits	9
1.4.1 Sametime Client Toolkits	9
1.4.2 Sametime Community Server Toolkit	11
1.4.3 The Sametime Directory and Data Access Toolkit	11
1.5 Why have Sametime-enabled applications?	12
1.6 What's new in the Sametime 3.1 APIs	12
1.6.1 Reverse proxy support	13
1.6.2 Multiplatform support in STLinks Toolkit	14
1.6.3 Status on login support	14
1.6.4 STLinks scalability upgrade	15
1.6.5 Fixed window size and place in chat dialogs	15
1.7 The structure of this redbook	16
Chapter 2. Setting up the development environment	21
2.1 Setting up the development environment	22
2.2 Installing the toolkits	22
2.2.1 Sametime Software Development Kit (SDK) documentation	22
2.3 The Java Client and Community Server Toolkits	24
2.3.1 IBM WebSphere Studio Application Developer 5.0	26
2.3.2 Borland JBuilder	31
2.3.3 Sun JDK	35
2.3.4 Installing the C++ Toolkits	37

2.3.5 Microsoft Visual C++	39
2.4 Installing the COM Toolkit	42
2.4.1 Microsoft Visual Basic	43
Part 2. Sametime enabling applications	47
Chapter 3. Sametime Bots	49
3.1 What is a bot?	50
3.2 Developing bots	51
3.2.1 Creating a Sametime session	52
3.2.2 Logging in to the community	53
3.2.3 Registering the message type	54
3.2.4 Listening for incoming messages	55
3.2.5 Responding with logic	55
3.3 Bot examples	56
3.3.1 The Echo Bot	56
3.3.2 FAQ Bot	61
3.3.3 Translator Bot	74
3.4 Enhancing the bot framework	79
Chapter 4. Web services	83
4.1 Overview of Web services	84
4.1.1 What is a Web service?	84
4.1.2 Web service fundamentals	84
4.2 Sametime Web services	85
4.3 Building the UserStatus application	86
4.3.1 The UserStatus application	86
4.3.2 Running the UserStatus application	94
4.4 Creating the UserStatus Web service	97
4.5 Deploying the UserStatus Web service	114
4.6 Testing the Sametime Web service	122
Chapter 5. Chat Logging/DDA Toolkit	127
5.1 Overview of the Chat Logging SPI and DDA Toolkit	128
5.1.1 Regulatory compliance	128
5.1.2 Collaborative commerce rules and workflow	129
5.1.3 Corporate and public sector governance	129
5.1.4 Appropriate use review	130
5.2 Developer considerations	130
5.2.1 Modes	131
5.2.2 Distributed environments	131
5.2.3 Synchronous and asynchronous implementations	131
5.3 Toolkit examples	132
5.4 Customizing and building a Chat Logger	136

Chapter 6. Sametime and workflow	139
6.1 Using Sametime within a workflow	140
6.2 The scenario	140
6.3 The AnnouncementSender application	141
6.3.1 The AnnouncementSender agent	141
6.4 The AnnouncementSender Web service	148
6.5 Sending rich text announcements with STLinks	153
6.6 Integrating with Microsoft Excel	155
6.6.1 Enabling Microsoft Excel for Web services	156
6.7 Summary	161
Chapter 7. BuddyList service	163
7.1 The BuddyList service advantages	164
7.1.1 The Sametime buddylist attribute structure	164
7.2 Overview of the Extended Live Names sample	166
7.2.1 Accessing the sample	166
7.2.2 Sample functionality overview	167
7.3 Setting up the development environment	171
7.4 Loading the user buddylist after successful login	172
7.4.1 Working with the BuddyList service	172
7.4.2 Handling service available/unavailable events	174
7.4.3 Loading the user buddylist	175
7.5 Handling a load failure	177
7.6 Keeping the buddylist updated	178
7.6.1 Adding a new user to a private group	178
7.6.2 Removing a person from the list	180
Chapter 8. Places and Place awareness	183
8.1 Key concepts within Places architecture	184
8.1.1 What are Places and why use them?	184
8.1.2 What are sections and why use them?	184
8.1.3 What are activities and why use them?	184
8.1.4 What are attributes and why use them?	185
8.2 Scenario	185
8.3 Application overview	185
8.3.1 Applying Places to the context of the scenario	187
8.3.2 Applying sections to the context of the scenario	187
8.3.3 Applying activities to the context of the scenario	189
8.3.4 Applying attributes to the context of the scenario	189
8.4 Setting up and running the application	189
8.4.1 Assign users to sections and the panel	190
8.4.2 Download and detach files	190
8.4.3 Environment variables	190

8.4.4	Sametime server preparation	191
8.4.5	Start the server application	191
8.4.6	Start the client applications	192
8.4.7	Understanding the client-side	192
8.4.8	Using the Applet GUI	192
8.4.9	Entering the Panel Discussion Place	195
8.4.10	Getting a reference to the activity	196
8.4.11	Getting references to sections	196
8.4.12	Entering the Queue	197
8.4.13	Sending a message to the panel.	199
8.4.14	Leaving the Queue	201
8.4.15	Logging in as a panel member	202
8.5	Building the server-side.	203
8.5.1	Logging in as a server application.	203
8.5.2	Creating a persistent Place and Place characteristics	206
8.5.3	Responding to a request for the activity	208
8.5.4	Monitoring the Place	209
8.5.5	Managing the Queue.	212
8.5.6	Receiving, translating, and responding to questions.	213
8.6	Summary	217
Chapter 9.	Sametime Links	219
9.1	Overview of Sametime Links.	220
9.2	Deployment considerations.	220
9.2.1	Size is important	220
9.2.2	Platform support	221
9.2.3	Working with anonymous users	222
9.3	Enabling live names in a Web page	223
9.4	Sametime Links directory overview.	224
9.4.1	Directory contents	224
9.4.2	Understanding the HTML files.	226
9.5	Building an interactive Web site	244
9.5.1	Provide the customer with a branded ST Links experience	245
9.5.2	Taking it further	255
9.6	Using Sametime Links with bots	256
9.6.1	How does it differ?	256
9.6.2	Changing the chatWindow.html	257
9.7	Adding menu options to Sametime Links	259
9.7.1	The sample pop-up menu	260
9.7.2	Starting with a Sametime Links-enabled page	262
Chapter 10.	Sametime-enabling portlets.	273
10.1	IBM WebSphere Portal Server overview.	274

10.1.1 IBM WebSphere Portal overview	274
10.1.2 Portlets overview	276
10.2 Versions of WebSphere Portal and Sametime	277
10.2.1 Included collaborative portlets	278
10.2.2 Collaboration Center for WebSphere Portal	279
10.3 Collaborative Components approach	281
10.3.1 Using WebSphere Portal Collaborative Components	281
10.3.2 Overview of the Collaborative Components API	282
10.3.3 Technical overview of the Collaborative Components	283
10.3.4 Adding Sametime collaboration to a basic portlet	285
10.3.5 Adding awareness to the JSP	286
10.3.6 Adding more Sametime functionality	289
10.4 STLinks API approach to enabling portlets	292
10.4.1 Why use the STLinks API	292
10.4.2 Enabling the portlet using STLinks	292
10.4.3 Conclusion	294
Chapter 11. Customizing the Online Meeting Center	297
11.1 Typical reasons for branding the Meeting Center	298
11.2 Branding the Meeting Center	298
11.2.1 Changing the page header	301
11.2.2 Changing the look	302
11.2.3 Using a Java Server Page (JSP) front-end	305
11.3 Meeting summary e-mail	305
11.3.1 Overview	306
11.3.2 Behind the scenes	309
11.3.3 Implementing the e-mail notification feature	312
11.3.4 Design changes	314
11.3.5 Validating the changes	326
11.4 Summary	331
Chapter 12. Ideas for customization and integration	333
12.1 Why customize and integrate?	334
12.2 Using Sametime to send data	334
12.2.1 The RichTextBot Sametime Bot	334
12.2.2 The RichTextClient Sametime applet	337
12.3 Alternative approaches to Single Sign-On (SSO)	341
12.3.1 The TokenGenerator servlet	342
12.4 Enabling Active Server Pages (ASP) with Sametime Links	351
12.4.1 What are IIS and ASP?	352
12.4.2 Directories	352
12.4.3 Logging in using the Token Generator	352
12.4.4 How it works	352

Part 3. Bringing it together.	357
Chapter 13. Visioning scenario: Sametime enterprise integration.	359
13.1 The scenario	361
13.2 Business drivers and requirements	361
13.2.1 Relationships between the call center and the departments.	362
13.2.2 Solution description.	366
13.3 Architectural considerations	367
13.3.1 Software components	368
13.4 Phased approach toward building the solution	369
13.5 Phase 1: Implementing the infrastructure	370
13.5.1 Implementing an infrastructure for instant collaboration	370
13.5.2 Customizing the meeting center look and feel	373
13.6 Phase 2: Expanding Sametime features.	378
13.6.1 Enabling people awareness into existing applications	379
13.6.2 Allow offline message delivery	383
13.6.3 Implement workflow using Sametime awareness feature	390
13.7 Phase 3: Expanding to outside world	392
13.7.1 Provide users with self service tools	393
13.7.2 Provide customers ability to call for online support.	400
13.7.3 Track user activity and provide active call center behavior.	408
13.7.4 Maintain logging of online customer conversations	411
13.7.5 Allow educational sessions with customers	413
13.7.6 Implementing multilanguage educational sessions.	417
13.8 Phase 4: Future planned enhancements	422
13.8.1 Consolidating the tools interface.	422
13.9 Conclusion.	425
Part 4. Appendixes	427
Appendix A. Visualizing Sametime	429
Overview	430
Architecture	430
Outline.	430
Visualizing Sametime client	432
Implementation.	434
GUI Design	434
Web services.	435
IM session data aggregation.	439
Outcome.	452
Trial program.	452
Results	453
Concerns.	453
Future improvements	454

Lessons learned	454
Appendix B. Online customer support application example	455
Overview	456
Sametime functionality supporting this application.	457
The customer applet.	458
Customer UI	458
Customer attributes.	460
Customer state machine	460
The agent applet	461
Agent UI	461
Agent attributes.	463
Agent state machine	463
Customer/Agent handshake	464
Helper classes	465
Back-end classes	465
Appendix C. Additional material	467
Locating the Web material	467
Using the Web material	468
How to use the Web material	469
Related publications	471
IBM Redbooks	471
Other publications	472
Online resources	472
How to get IBM Redbooks	473
Help from IBM	474
Index	475

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

alphaWorks®
ibm.com®
iSeries™
zSeries®
AIX®
AS/400®
Domino™
Domino Designer®

Everyplace®
Informix®
Intelligent Miner™
IBM®
Lotus Discovery Server™
Lotus Notes®
Lotus®
Notes®

OS/390®
OS/400®
QuickPlace®
Redbooks™
Redbooks (logo) ™
S/390®
Sametime®
WebSphere®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Preface

In this IBM Redbook, we explore Sametime® as a development platform, illustrating how to extend the functionality of Sametime beyond its more commonly known instant messaging and e-meeting hosting features.

We provide a detailed technical discussion and examples for building and integrating Sametime real-time collaborative capabilities, presence awareness, and Web conferencing capabilities into both new and existing applications. In depth discussions and code examples are provided for topics such as building custom bots, leveraging Sametime functionality through Web Services, and integrating Sametime into workflow applications. Additionally, this redbook reveals how you can customize (brand) your organization's Sametime Meeting Center to more closely match the company's identity. Finally, very thorough analysis is given to the topics of Sametime Links and the Sametime Places architecture. Several samples of Sametime applications, with source code, are included.

Ultimately, we hope this redbook will help you better appreciate how your organization may benefit by more effectively leveraging Sametime.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Cambridge, Massachusetts.

John Bergland is a project leader at the International Technical Support Organization, Cambridge Center. He manages projects that produce Redbooks about Lotus® Software products. Before joining the ITSO in 2003, John worked as an Advisory IT Specialist with IBM® Software Services for Lotus (ISSL), specializing in Notes® and Domino™ messaging and collaborative solutions.

John Barrow is a software developer for PriceWaterhouseCoopers (PwC) in Manchester, UK. He has five years of experience in the IT field, four of them with PwC. He holds a degree in Geology from the University of Edinburgh and is a PCLP Application Developer and a CLP System Administrator. His areas of expertise include Domino Web development, Lotus Team Workplace, and Lotus Instant Messaging/Web Conferencing.

Jonas Covey is a Senior I/T Specialist with IBM Software Services for Lotus. He has nine years of experience in the IT field, five of which have been with IBM. His areas of expertise include Lotus Team Workplace, Lotus Instant Messaging/Web Conferencing, and Web development.

Washington Cabral is a IBM IT Architect in Brazil. He has nine years of experience in the Lotus technologies field. He is a member of IBM Software Services for Lotus Americas Architects Council, a Certified Lotus Instructor, and a Certified Lotus Professional. His areas of expertise include general architectural solutions on middleware integration and messaging deployment, messaging migration, and Sametime planning and deployment. He has written extensively about integrating the Sametime solutions described in this redbook into enterprise environments, as well as about the techniques involved in Sametime-enabling portlets on IBM WebSphere® Portal.

Carl Tyler is Chief Technology Officer for Instant Technologies, a company that specializes in building advanced Real Time Collaboration applications, such as the award winning Instant TeamMessenger for Microsoft® Outlook. Carl heads the technical direction of Instant Technologies. He has over 15 years of experience with collaboration software, including over nine years at Lotus Development and two-plus years at IBM Development UK. While at Lotus, he was Worldwide Knowledge Management Manager and Lotus Notes® R5 Marketing Manager.

Yafit Sami is the member of the Sametime development group working in the IBM Haifa Software Lab in Rehovot, Israel. She has been with the Sametime development team for seven years, serving as the team leader of the group that developed the Sametime C++/COM Toolkits and is currently the team leader of the Sametime Links API. She is also involved in the development of the Sametime mobile solution. She has extensive experience working with developers using the Sametime Toolkits, helping to answer questions and consulting on design solutions.

Rob Novak is president of SNAPPS, headquartered in Overland Park, Kansas. SNAPPS is the winner of the 2002 Lotus Beacon Award for Rising Star. Rob and his team focus on customization, global implementation planning and execution, and developing tools for QuickPlace, Sametime, and Domino for global corporations. A ten-year Notes and Domino veteran, Rob frequently speaks on QuickPlace, Sametime, Domino, and Lotus Workplace at conferences and seminars. He recently co-authored the new Lotus certification exams for QuickPlace, Sametime, and Notes/Domino 6. Rob is an IBM Certified Advanced Application Developer, holds two Masters degrees, and a Doctorate in business.

Thanks to the following people for their contributions to this project:

Jenifer Servais
International Technical Support Organization, Rochester Center

Amy Reuss Caton,
Lotus Software, IBM

Brian White
Lotus Software, IBM

Darren M. Shaw
IBM Emerging Technologies Group, IBM UK

Vince Fertig
IBM Software Services for Lotus

William Tworek
International Technical Support Organization, Cambridge Center

Bob Balaban
President, Looseleaf Software

Peyton McManus
President, Instant Technologies.

The following people also proved as valuable resources for questions, discussions about Sametime, and for reviewing the content.

- ▶ Haim Schneider, IBM
- ▶ Marjorie Schejter, IBM
- ▶ Dave Stokes, IBM
- ▶ Stan Logan, IBM
- ▶ Leonard Rania, IBM
- ▶ Jacques Perrault, IBM
- ▶ Baan Al-Shibib, IBM
- ▶ Mike Ebbers, IBM
- ▶ Benjamin E Maynard, IBM
- ▶ Mark Feinman, IBM

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JLU Mail Station P099
2455 South Road
Poughkeepsie, New York 12601-5400



Part 1

Introduction

This part provides an overview of Sametime, its functionality, and the available toolkits provided with the Sametime 3.1 SDK. It also discusses how to set up several common integrated development environments (IDEs) in order to begin developing with Sametime.



Overview of Sametime

IBM Lotus Instant Messaging and Web Conferencing (Sametime) is the market leading instant messaging and Web conferencing solution for business. Upon discovering the functionality and capabilities of Sametime, you will soon appreciate that it is much more than a stand-alone application or simple instant messaging tool. Sametime is ultimately a powerful platform for real-time enabling Web sites, Windows® applications, and Web applications. In this redbook, we explore Sametime as a development platform, illustrating how to extend the functionality of Sametime beyond its more commonly known instant messaging and e-meeting hosting features. Using Sametime and its available toolkits, this redbook provides illustrative examples of how to build and integrate Sametime functionality into custom applications. Ultimately, we hope this redbook will help you better appreciate how your organization may benefit by more effectively leveraging Sametime.

The primary target audience is developers, architects, and IT managers looking to further leverage the capabilities of Sametime. Please note that this redbook builds upon two existing Redbooks, *Working with the Sametime Client Toolkits*, SG24-6666, and *Working with the Sametime Community Server Toolkit*, SG24-6667.

In this chapter, we start by giving a brief overview of what Sametime really is. We provide a high level overview of the toolkits available for Sametime, referencing two previous IBM Redbooks, which provide a more in depth examination of the toolkit capabilities, and highlighting new capabilities provided with the Sametime

3.1 APIs. Finally, we describe the structure and content to be covered in the rest of the redbook.

1.1 Benefits and importance of collaboration

As a foundation for understanding the benefits of Sametime and how it may help your organization, it is important to acknowledge the importance of collaboration. Collaboration represents the simple act of working together to accomplish a common goal. Sametime allows for real-time collaboration.

Integrating collaborative services with business functions allows companies to gain a significant competitive advantage. Information is shared more effectively, communication is more efficient, and companies can make quicker, more informed decisions. More specifically, companies can shorten sales cycles, accelerate product development, generate more transactions, increase partner/customer retention, and expedite problem resolution. Ultimately, these collaborative capabilities provide competitive advantage in the marketplace and impact the bottom line.

1.2 What is Sametime?

Lotus Sametime is IBM's market-leading platform for real-time collaboration. People use it every day to share information, make faster decisions, and work together in real time. Lotus Sametime was engineered from its very beginning to be a tool for e-business, and that is why it is robust, secure, and manageable.

Lotus Sametime is based on three basic concepts:

- ▶ Presence/Awareness (seeing in advance whether a person(s) or application(s) is available to collaborate, share information, and/or take an action)
- ▶ Conversation (text-based or audio/video)
- ▶ Web conferences (application sharing and team whiteboarding)

Any organization that needs to increase its productivity, be more efficient, work faster, reduce travel budgets, integrate real-time collaboration into Web applications, or increase its agility will benefit from using IBM Lotus Sametime.

1.3 Sametime services

Sametime API functionality and accessibility is provided through Sametime services. Some of these services are isolated, while others are dependent upon each other. By providing functionality and accessibility through services, the Sametime API remains extendable, while also providing the following key benefits:

- ▶ Future Sametime versions can add services and still work with older applications and toolkits.
- ▶ You can exchange the toolkit for a newer version without changing your application, so long as the existing services have not changed the API.

Note that not all services are supported by each of the Sametime Toolkits. Section 1.4, “Overview of the Sametime 3.1 Toolkits” on page 9 provides additional detail on the services provided with each toolkit.

Sametime services fall into three areas:

- ▶ **Community services:** These services include awareness, instant messaging, instant meetings, and chat. A buddylist makes Sametime users aware of who is available (and who is online but unavailable) to receive an instant message or participate in a chat with one or more people. All chat content is encrypted.
- ▶ **Online Meeting services:** These services include a shared whiteboard and the ability to share programs and documents online. Sametime also offers a server-based Meeting Center where users can schedule online meetings in advance and store agendas and other meeting materials. Additionally, audio and video can be directly integrated into online meetings.
- ▶ **Customization and Integration services:** Sametime also provides a comprehensive API that enables customers to easily integrate real-time collaborative capabilities into other applications, such as e-commerce sites, help desks, and training/information delivery applications like Sales Force Automation.

1.3.1 Real-time collaboration: community services

Online, real-time collaboration is most effective when it occurs spontaneously, just like the hallway encounter. But like a face-to-face encounter, you need to be aware of the opportunity to interact. Sametime recognizes this fact and incorporates the ability to tell the server your availability. A user can tell the server whether they are online, away from their computer, to please not disturb, or to let people know that they are on a mobile device. The awareness capabilities of Sametime help make spur-of-the-moment, online conversations as natural, convenient, and worthwhile as a hallway chat. In certain situations where text chat may not be enough, Sametime’s audio and video support allows a

much more personal and productive tool to complement a typical text chat. Sametime makes users aware of opportunities for online interaction via a sophisticated buddylist, used to identify which members of a community are online and whether they are available to interact. Once users are aware of who is online, they can initiate interaction simply by sending an instant message. Sametime gives each user full control over their availability. Levels of participation include Active (online and available), Away (offline or otherwise unavailable) and Do Not Disturb (online but unavailable). Every user can also control their privacy list by specifying a list of users who can see whether or not they are online.

Sametime enables presence awareness and person to person conversation from either a contact list (buddylist) in one of the official Sametime clients (Sametime desktop client, Java™ connect client or Meeting room client), or from any other application where Sametime functionality has been built in. An example of the Sametime desktop client is shown in Figure 1-1.

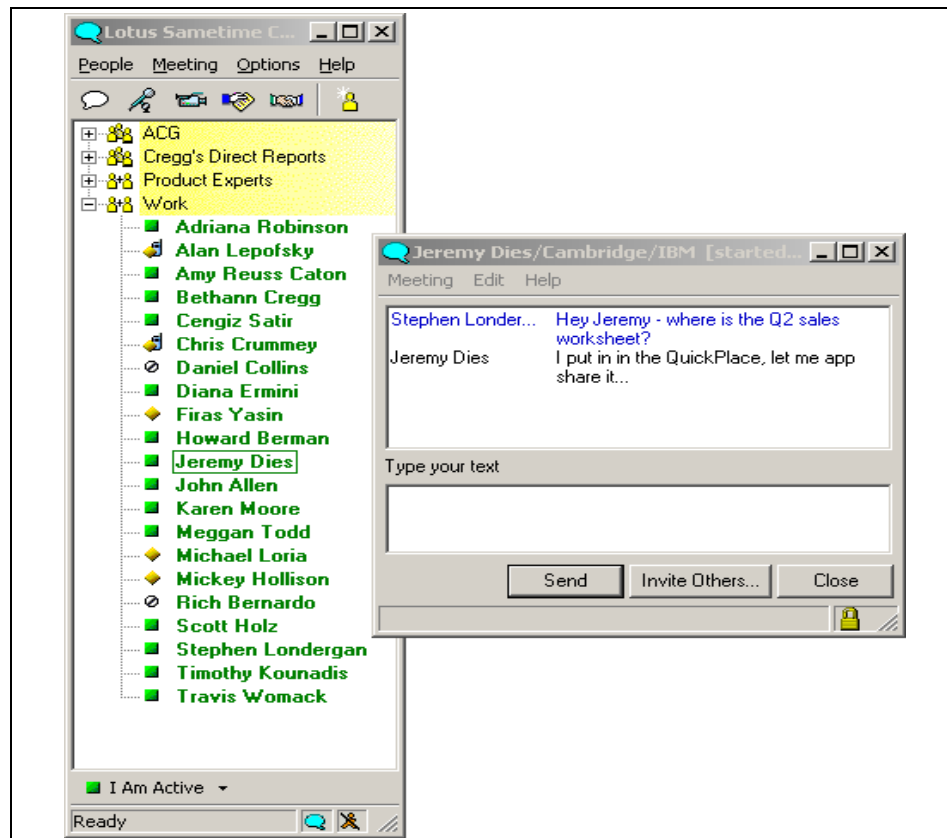


Figure 1-1 The Sametime buddylist

1.3.2 Sametime online meeting services

Sametime online meeting or conferencing services provide the ability to share objects (such as desktop applications, presentations, documents, and drawings) online. Users can schedule an online meeting in advance, or move directly from an instant message to a screen-sharing or whiteboard session, such as the one shown in Figure 1-2.

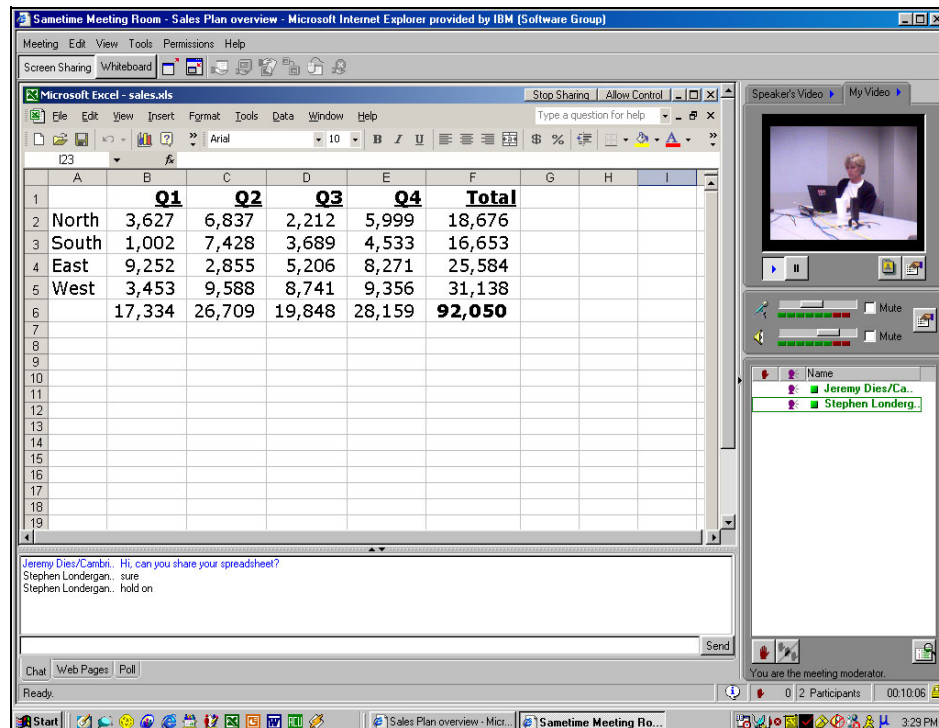


Figure 1-2 Sametime client for application sharing and online meetings

Sametime allows any user to share any application from his or her desktop, such as word processors, spreadsheets, and project management software. If necessary, a user can also share their entire desktop, enabling a remote participant to control their machine. Other participants are not required to have the same software in order to participate and see what is being shared. When appropriate, users can also pass control of the application or the desktop back and forth as necessary; the initiator can reassert control at any time. Sametime's shared whiteboard is the online equivalent of a typical whiteboard in an office or classroom. Users can draw on it, show presentations, and annotate documents on it. Sametime also converts popular file types into "pages" for convenient display during whiteboard sessions.

1.3.3 Sametime customization and integration services

In addition to secure instant messaging and online meetings, Sametime offers developers a comprehensive set of tools, components, and APIs to integrate real-time collaboration capabilities with other applications. The APIs are exposed via toolkits that enable you to write client applications in Java, C++, COM or HTML, and JavaScript. In addition, there is a Java Toolkit for server-side applications.

1.4 Overview of the Sametime 3.1 Toolkits

This section provides a brief, high level review of the Sametime Client Toolkits, the Sametime Community Server Toolkit, and the Directory and Data Access Toolkit. While this section is intended to provide an overview of the available toolkits for Sametime development, the functionality has been discussed in much greater detail in the following two Redbooks, *Working with the Sametime Client Toolkits*, SG24-6666, and *Working with the Sametime Community Server Toolkit*, SG24-6667. Developers should reference these books to gain a much more in-depth understanding of toolkit comparisons, as well as detailed techniques and methods for using them.

1.4.1 Sametime Client Toolkits

The Sametime Client Toolkits provide the ability to Sametime-enable software applications in both Java and Windows environments. In this section, we present an overview of the following Sametime Client Toolkits:

- ▶ Sametime Links Toolkit
- ▶ Sametime COM Toolkit
- ▶ Sametime Java Toolkit
- ▶ Sametime C++ Toolkit

Sametime Links Toolkit

The Sametime Links Toolkit is an easy-to-use, lightweight (~20K applet) ADA-compliant Toolkit that allows Web developers to enable HTML and JavaScript pages and applications with Sametime "live names." The toolkit is HTML-based and customizable with no installation necessary. It can be used in any Web development environment, as there is no Java programming required. For example, the developer can turn existing names into Sametime Links with the addition of a few lines of HTML, without affecting the page layout.

The Sametime Links Toolkit allows you to add awareness and messaging to your Web pages and applications, a distinct e-commerce advantage. Compatibility with additional operating systems and browser versions has also been enhanced. Refer to Table 1-1 on page 14 for a list of compatible operating systems and browser versions.

Sametime COM Toolkit

The Sametime COM Toolkit provides the basic Sametime community services, delivered as a standard DLL with interfaces that represent the services provided. The toolkit is easy to use and can be used to enable applications with Sametime in any development environment that supports COM, such as Visual Basic/VBA (Microsoft® Office.)

While the Sametime COM Toolkit is relatively limited in its functionality compared to the Java Toolkit, it does provide access to any environment that supports COM objects. For example, if you want to develop Sametime awareness for Microsoft Office products, then the Sametime COM Toolkit is the answer. The toolkit also supports the OLE Automation technology by providing a TLB file as well as a DLL to support importing all interface names. By using your own development tools, you can create your own user interface on top of the Sametime services.

Sametime C++ Toolkit

The Sametime C++ Toolkit is an object-oriented, modular, and thread-safe API. This Toolkit is a collection of components that allow developers to enhance Windows applications with community services, such as awareness and instant messaging. Using the Microsoft Visual C++ Developer Studio, developers can quickly and easily Sametime-enable any Win32 or MFC-based Windows application.

You can use this toolkit if you want to develop applications native to the Windows environment. Unlike the Java Toolkit, the Sametime C++ Toolkit does not have a user interface (UI) component. If you create an application in which a user interface is expected, then you will have to develop your own interface.

One very popular example of an application developed using the Sametime C++ Toolkit includes NotesBuddy from IBM's alphaWorks® (<http://www.alphaworks.ibm.com>). NotesBuddy is a lightweight tool for Lotus Notes and Internet (POP3) mail that integrates instant messaging and buddy status with e-mail to produce a single messaging tool. NotesBuddy can access the Sametime server and provide instant messaging and buddy status. Chats or e-mail can be initiated anywhere a buddy name is displayed and then saved to folders or forwarded to others. Rich text and graphics (such as smiley faces and

Web graphics) can also be exchanged when chatting with other NotesBuddy users. For more information on this tool, visit:

<http://www.alphaworks.ibm.com/tech/notesbuddy>

Sametime Java Toolkit

The Sametime Java Toolkit is a collection of building blocks or components that developers use to build applications that leverage the functionality and services provided by Lotus Sametime. These Toolkit components can be used in any standard development environment that supports JDK 1.1.8 or above.

The Sametime Java Toolkit is an object-oriented, modular, and thread-safe API. While somewhat larger than the toolkits described above, the Sametime Java Toolkit provides all the core Sametime community and meeting services and UI components. Developers wanting to embed Sametime-based services and functionality in Web applications can use the Sametime Java Toolkit. For example, a developer could use the Java Toolkit to build a facility for live customer-service-style help in an online marketplace, or perhaps build awareness into a knowledge management application, or bring application sharing into an e-business application.

1.4.2 Sametime Community Server Toolkit

The Sametime Community Server Toolkit is one member of a comprehensive, Java-based application SDK that developers can use to embed real-time capabilities into e-business applications. This server toolkit is a collection of components used by developers to build applications that affect the functionality and services provided by a Sametime server. These components can be used in any standard development environment that support JDK 1.1

The Sametime Community Server includes many Server Applications, each of which provide some of the Sametime functionality. The Sametime Community Server Toolkit provides the ability to write new server applications, thus providing new services to the Sametime community. The toolkit is easy to use and can be used to enable server applications with Sametime in any development environment that supports Java.

1.4.3 The Sametime Directory and Data Access Toolkit

The Sametime Directory and Data Access Toolkit allows you to customize Sametime to handle the following tasks in your Sametime deployment:

- ▶ Chat logging: This server-side component allows transcripts of all chats to be stored in a set location, for example, in a specific database. This SPI allows

the administrator to determine where and how the chats are stored, by developing a DLL or shared library that implements the specified SPI.

The Sametime Chat Logging SPI is implemented as a DLL for the Windows® operating system supported by the server and as a shared library for IBM® AIX® and Solaris. The chat logging sample in the Toolkit is currently implemented only for the Windows operating system with the Microsoft® Visual C++ 6.0 Developer Studio SP5; however, AIX and Solaris developers may use the same source files in adapting the sample for use in these environments.

- ▶ Token authentication: Sametime supports two types of authentication tokens out-of-the-box:
 - Proprietary Sametime token
 - LTPA token supported by Domino and WebSphere

This server side component allows you to customize Sametime for a different kind of token generated in your deployment, by developing a DLL or shared library that implements the specified SPI.

1.5 Why have Sametime-enabled applications?

Instant messaging and application sharing technology is essential for real time collaboration, but having a contact/buddylist as a stand-alone application limits the user's work process. In order to initiate contact, a user must stop what they are doing and go outside of a specific application to communicate. Using Lotus Instant Messaging technology and the available Sametime Toolkits, it is possible to build custom applications which integrate presence awareness and chat capabilities directly into the context of the application. This enables users to communicate and collaborate *directly from within the context* of the application where they are working.

1.6 What's new in the Sametime 3.1 APIs

Sametime 3.1 is a minor feature release that has two main objectives. One goal of release 3.1 is to address and fix as many problems as possible that have been reported by customers who use earlier versions of Sametime. The second objective is to add certain key features that have been added due to strong customer demand.

For the list of problems that have been fixed, you can go to the Sametime fix list database on the Web at:

<http://www-10.lotus.com/ldd/stfixlist.nsf>

This following sections discuss the new features that were added in Sametime 3.1 that affect the various Sametime APIs. For the full list of new features (both end-user feautres, as well as features impacting the APIs, please see the “What’s new” section of the *Sametime 3.1 Release Notes*, found at:

<http://www-12.lotus.com/ldd/doc/sametime/3.1/strn31.nsf/NNew?OpenNavigator>

1.6.1 Reverse proxy support

A Sametime 3.1 server can be deployed behind a reverse proxy server. End users can connect to the Sametime server through the reverse proxy server. The STLinks and Java Toolkit support reverse proxy environment in Sametime 3.1.

Reverse proxy support in STLinks Toolkit

There are two new variables declared in the stlinks.js run-time file:

- ▶ `ll_proxyName`: The reverse proxy name
- ▶ `ll_AffinityId`: The Sametime server affinity ID

These variables are commented out with a default value and should be modified by the administrator when using a reverse proxy environment. The administrator should change the default values to the real ones.

After setting those variables, STLinks Toolkit is ready to support a reverse proxy environment. There are no changes in the API resulting from this new reverse proxy support and no additional required code by the developer.

Reverse proxy support in Java Toolkit

There is no default support for a reverse proxy environment in the Java Toolkit. However, a new API has been added to the Java Toolkit in order to allow applications to take advantage of this support. The new API concerns the connection classes like `HTTPConnection`, `PollingConnection`, and so on. In the `com.lotus.sametime.core.util.connection` package, the Sametime development team has added constructors that accept a URL parameter versus host/port in the older constructors.

For example, we added a new `HttpConnection` constructor:

```
public HttpConnection(URL hostUrl, long timeout
```

Note: This new API (constructors) can be used for any other purpose than Reverse Proxy support. It is important to understand that they are not implemented directly for the purpose of Reverse Proxy behavior.

1.6.2 Multiplatform support in STLinks Toolkit

STLinks Toolkit now has multiplatform support in Sametime 3.1. Platform support now includes the AIX, Solaris, and Linux platforms. Browser support now also includes Netscape 7 instead of Netscape 4.7. Operating system and browser compatibility is shown in Table 1-1.

Table 1-1 Operating system and browser compatibility for Sametime Links Toolkit

Operating system	Browser version
Windows	IE 5 and higher, Netscape 7
AIX	Netscape 7
Solaris	Netscape 7
Linux	Netscape 7

1.6.3 Status on login support

A new feature in Sametime 3.1 is the Status on Login feature. This feature gives you the ability to choose your status upon initially logging in to the community. In all the previous Sametime releases, whenever the user logged in to the community (from any client), his status automatically became Active. The user had to change it manually, or have the client change it through an automated process. In some cases, this could cause an extra refresh for other users' contact lists, as well as create an unnecessary pop up alert dialog for other clients. An example of this scenario is as follows:

1. User 1 asks to be alerted about user 2 whenever he becomes Active. User 2 logs in and then the client automatically changes his status to an Away mode.
2. User 1 will get the alert that the user is Active and then will see him as an Away in the contact list. This type of scenario could be very confusing to users.

We can avoid this problem by using the new feature Status on Login. Using this feature tells the server to log in user 2 with status of Away, so user 1 will not get the alert. This feature has been added to the Java and C++ Toolkits.

Java/C++ Toolkits

There is a new method in the CommunityService:

- `setLoginStatus (bool forceChangeStatus, STUserStatus loginStatus):` Sets the status to be used in the next login try (does not affect current user status).

Community Server Toolkit

No work is needed; you will get this feature by using the latest Java Toolkit.

1.6.4 STLinks scalability upgrade

As part of the Sametime 3.1 release, scalability improvements have been made to more easily support up to 300,000 concurrent Sametime Links clients.

Sametime Links is installed out of the box as part of Sametime 3.x and can easily support, on a single server, several thousand concurrent Sametime Links users in addition to tens of thousands of regular Sametime clients, such as Sametime Connect. In order to support a very large number of Sametime Links clients, additional hardware is required. We measured a clustered architecture with up to 15 machines. Our results (see Table 1-2) show a linear scaling as we increased the number of machines. Overall, we demonstrated that Sametime could support 300K concurrent Sametime Links clients, with a high probability that more can be supported with additional machines.

Table 1-2 Scaling Sametime Links

Community hub machines	STLinks/Mux machines	Sametime Links client supported
1	1	25,000
1	2	50,000
1	3	75,000
1	4	100,000
2	8	200,000
3	12	300,000

Prior to the release of Sametime 3.1, customers and the development team experienced many issues getting to this level of scalability. Accordingly, these fixes are part of the Sametime 3.1 release.

1.6.5 Fixed window size and place in chat dialogs

This feature is enabled only for developers who use the Java Toolkit UI component.

End users can control the default location on their screen in which chat windows pop up when the user participates in a chat session. End users can also control the size of the chat window when it pops up on the screen.

Note: The administrator must enable this feature before it is available for end users.

1.7 The structure of this redbook

This redbook is divided into three parts:

- Part 1: Overview of Sametime

The current chapter is intended to provide the reader with an overview of Sametime and its capabilities, as well as outline the focus and content of the redbook.

In Chapter 2, “Setting up the development environment” on page 21, we examine how to set up and configure some of the more popular integrated development environments (IDEs) for Sametime application development. Developers often are interested in learning how to develop within Sametime, yet are simply not familiar with how to properly set up a development environment. This chapter provides helpful guidance.

- Part 2: Sametime-enabling Applications

Chapter 3, “Sametime Bots” on page 49 discusses the concept of Sametime Bots, what they are, and how to build them. It begins with a general discussion of what a bot is, then discusses the various ways that you can build a Sametime Bot using the available toolkits. We examine in detail a couple of bots developed using the Java Client Toolkit to show how they work and how you can create your own. Finally, it concludes by suggesting enhancements to the bot concept, a discussion around possible deployment scenarios, and additional ideas for potential bots.

Chapter 4, “Web services” on page 83 provides an overview of Web services and their usage with Sametime. This chapter illustrates how you can make online presence awareness and instant messaging a part of any Web services-enabled system. It provides a specific example to demonstrate how applications created with the Java Client Toolkit can be turned into Web services using the tools provided by WebSphere Studio Application Developer 5.0. Once developed, it explains how these Web services can then be deployed to an application server, such as WebSphere Application Server, and accessed by any Web services client.

Chapter 5, “Chat Logging/DDA Toolkit” on page 127 discusses the chat logging SPI. It discusses the importance of chat logging capabilities, a topic that is especially relevant given the recent changes within regulatory compliance requirements. Additionally, it illustrates which capabilities are possible using the provided SPI, while also discussing an example about how to create your own custom chat logging recorder.

Chapter 6, “Sametime and workflow” on page 139 discusses how Sametime can be used within a workflow application context. It illustrates how you can connect your application to Sametime and leverage the functionality that it offers. A number of different scenarios are provided to help expand your ideas about where you may wish to incorporate Sametime into existing business processes. One scenario, described in detail, illustrates how to enable a Domino workflow application with awareness and instant messaging capability. We then take this idea a step further and discuss how to utilize Sametime Web services in applications. We show a Web service that acts as an announcement service, allowing applications to send instant announcements to selected online users.

Chapter 7, “BuddyList service” on page 163 discusses the BuddyList service component. It builds upon both the “Basic Buddylist” sample provided with the Sametime Toolkits, and an earlier example provided with the C++ Toolkit, the “Extended Live Names” sample. The objective of the chapter is to illustrate how the BuddyList service provides developers with an easier alternative to the Storage service for loading a buddylist. The BuddyList service allows you to do the same actions with less coding and with no internal knowledge regarding the attribute format in the server.

Chapter 8, “Places and Place awareness” on page 183 examines Places architecture in extensive detail from both a client side and server side perspective. It reviews the concept of Places, sections, attributes, and activities, and discusses why it is advantageous to use them. Next, it addresses the key concepts of Places and Place architecture through an application example that implements a multi-lingual panel discussion by leveraging Places, sections, Place awareness, and features in the Sametime Community Server Toolkit.

Chapter 9, “Sametime Links” on page 219 provides an overview of the Sametime Links Toolkit and explains the usage and purpose of each file that is used by Sametime Links. A detailed approach to using Sametime Links is provided in an example of using Sametime Links on a Web site to build in visitor interaction.

Chapter 10, “Sametime-enabling portlets” on page 273, covers the topic of integrating Sametime functionality within a portal and portlets. It discusses two approaches to enabling portlets with presence awareness and instant messaging capabilities:

- Using the Collaborative Components API provided with WebSphere Portal
- Using Sametime Links Toolkits within a portlet

An example of each approach is provided.

Chapter 11, “Customizing the Online Meeting Center” on page 297 discusses techniques and ideas for customizing the online meeting center to more

closely have the meeting center match a company's intranet and overall corporate identity. In addition to discussing typical reasons why people brand the Meeting Center, this chapter includes an example of branding the Meeting Center and how to add a feature to send an e-mail notification whenever a meeting is created, updated, or deleted.

Chapter 12, "Ideas for customization and integration" on page 333 begins to further explore the things that are possible with Lotus Sametime when you start to broaden your horizons past the idea of traditional instant messaging. Although the Sametime Connect client is not customizable, with the power of the Sametime Toolkits you can build very powerful systems that can interact with each other through the instant messaging infrastructure. As an example, this chapter illustrates how to send binary data between a custom Java applet client and a custom bot which checks and reports on an item's status from within a part inventory system. Finally, this chapter also examines possible alternative approaches to Single Sign On (SSO) Authentication. This information may prove especially helpful for anyone wishing to integrate Sametime into an environment that does not contain WebSphere or Domino and can not use Lightweight Third Party Authentication (LTPA) tokens.

- ▶ Part 3: Bringing it all Together: Introducing a business context

Finally, Chapter 13, "Visioning scenario: Sametime enterprise integration" on page 359 shifts our focus a bit in order to provide a business context and help an IT manager or executive understand the different possibilities for integrating Sametime into the enterprise. The objective of the chapter is to provide a business and architectural context that illustrates how many of the code examples can be incorporated into a "real world" business scenario. All technologies and features described in earlier chapters are tied together in a logical way in order to provide the reader with the necessary information for integrating Sametime into a business scenario.

- ▶ Part IV - Appendices

Appendix A, "Visualizing Sametime" on page 429 provides a description and code samples from *Visualizing Sametime*, an application written by Darren M. Shaw of IBM's Emerging Technologies group. Visualizing Sametime (VS) was designed as a proof of concept system to show that social network graphs and analysis could be generated from instant messaging traffic. Darren provides details about the architecture, technologies, and implementation methods used in building this application. This application also generated valuable lessons that are applicable to any Sametime Developer.

- ▶ Appendix B, "Online customer support application example" on page 455, discusses a larger and more complete Sametime application. This application provides an alternate means of handling customer support through live chats between a customer and customer support agents. The goal of this appendix

is to illustrate an overview of the architecture of the application, while also highlighting some of the important design decisions and their potential benefits to the application.



Setting up the development environment

To build Sametime-enabled applications, you need a development environment that is configured correctly. This chapter provides a guide to setting up some of the more popular integrated development environments (IDEs) for Sametime application development.

The Sametime Toolkits that will be discussed are:

- ▶ Java Client Toolkit
- ▶ Java Community Server Toolkit
- ▶ C++ Toolkit
- ▶ COM Toolkit

The IDEs that will be discussed are:

- ▶ IBM WebSphere Studio Application Developer 5.0
- ▶ Borland JBuilder
- ▶ Sun JDK
- ▶ Microsoft Visual C++ 6.0
- ▶ Microsoft Visual Basic 6

2.1 Setting up the development environment

The following sections show how to set up some of the most common IDEs to use the various Sametime Client and Server Toolkits.

Attention: This chapter assumes you have correctly set up your chosen IDE. It does not give details on how to install any IDE, just how to enable them to be used for Sametime development.

2.2 Installing the toolkits

If installed on the Sametime server, all the Sametime Toolkits can be downloaded from the toolkits page.

1. Navigate to the home page of the Sametime server (http://<your_Sametime_server>/stcenter.nsf).
2. Click on the SDK link at the bottom of the page.

Important: The toolkits must be installed after you have installed the Sametime server; the server installation itself does not automatically install the toolkits as well.

2.2.1 Sametime Software Development Kit (SDK) documentation

Once the Sametime Software Development (SDK) toolkits have been installed, there are a variety of online documents, code samples, and online tutorials that can be accessed on the Sametime server. A tutorial can be downloaded from a Sametime server that has the Sametime Software Development Kit (SDK) installed. You should be able to access a listing of the toolkits on your Sametime server using the following URL:

<http://<sametime server name>/sametime/toolkits/>

Figure 2-1 on page 23 illustrates a listing of the toolkit links available from the SDK page of a Sametime server with the toolkits installed.

The Sametime Software Development Kit (SDK)

Here you can download toolkit APIs, documentation, and samples in order to start adding Sametime capabilities to your applications. The following Toolkits are currently available:

Sametime 3.1 Toolkits	Description
Java Toolkit	Provides Sametime Community and Meeting services, allowing Java developers to embed real-time collaboration technology into new or existing applets, applications, and web sites
C++ Toolkit	Provides a collection of static library components that allow developers to enhance Windows applications with Community Services, such as awareness and instant messaging
COM Toolkit	Provides a COM object with basic Sametime Community Services (login, awareness, and instant messaging) for Visual Developers
Sametime Links Toolkit	A HTML/JavaScript API for adding "live names" to web applications or web pages with a few lines of HTML
Community Server Toolkit	Server API to develop server applications, get events, change attributes, and administer places
Directory and Database Access Toolkit	C APIs to modify Sametime for existing directories and databases
Chat Logging API	Allows the administrator to determine where and how logged chats are stored, by developing a DLL (or shared library) that

Figure 2-1 Listing of the toolkit links from a Sametime Server with the SDK installed

From within each of the toolkit URLs, you will find helpful documentation, samples, and tutorials relevant to the specific toolkit. For example, each of the documents shown in the list below are contained within the SDK on a Sametime server:

- ▶ *Sametime 3.0 Directory and Database Access Toolkit Developer's Guide*
- ▶ *Sametime 3.1 Administration Guide*

- ▶ *Sametime 3.1 C++ Developer's Guide*
- ▶ *Sametime 3.1 Installation Guide*
- ▶ *Sametime 3.1 Java Toolkit Developer's Guide*
- ▶ *Sametime 3.1 Release Notes*
- ▶ *Sametime Links 3.1 Toolkit Developer's Guide and Reference*
- ▶ *Sametime Links JavaScript API Reference*

2.3 The Java Client and Community Server Toolkits

The Java Client and Server Toolkit binaries can be downloaded by following the links on the toolkits page to the Java Toolkit and Community Server Toolkit, respectively.

The Client Toolkit consists of a number of files; Table 2-1 provides a description of the major ones.

Table 2-1 Major Java Client Toolkit binaries

File	Description
STComm.jar (and STComm.cab)	Includes the classes that provide the Community Services functionality, such as people awareness, instant messaging, Places, and Place-based awareness.
CommRes.jar (and CommRes.cab)	Includes text and graphic resources used by some of the UI elements in the other toolkit packages.
STMeeting.jar (and STMeeting.cab)	Includes the classes that provide the Meeting Services functionality. Meeting Services provide real-time collaboration services for meetings. Most Meeting Services rely on Community Services and provide AWT components to render the different activities they represent.

The rest of the binary files include packages to provide support for broadcast clients in Microsoft Internet Explorer or Netscape, as well as Windows platform support for application sharing. Refer to the *Sametime 3.1 Java Toolkit Developer's Guide* (see 2.2.1, "Sametime Software Development Kit (SDK) documentation" on page 22 for information on finding this guide) for more information about the contents of the toolkit binaries.

The Community Server Toolkit is an extension to (a superset of) the Java Client Toolkit, and it is written using the same underlying architecture. The full Java Client Toolkit is included in the Server Toolkit.

The Server Toolkit consists of two binary files, as described in Table 2-2.

Table 2-2 Java Server Toolkit binaries

File	Description
stcommsrvrtk.jar	Includes all the classes for the Java Client and Server Toolkits
commres.jar	Contains resources used by the toolkits

You can view the Client Toolkit and the Server Toolkit as one large toolkit, which is divided into two logical parts. The Server part enables its users to add application logic on the server side by writing new server components. The Client part gives the ability to transfer this logic to the user.

This means that there are, for example, two different packages for Places functionality: one for the Client and one for the Server. For example:

```
com.lotus.sametime.places  
com.lotus.sametime.placessa
```

Both of these packages contain classes with names like Place, Section, PlaceMember, and so on. If you import both packages in your class, you have to use the fully qualified name to refer to the specific class, for example:

```
com.lotus.sametime.places.Section
```

If you try and use the short form, such as Section, you will get an error message about ambiguous types. However, you should normally only need one of the packages. You use the server-side Places package for server applications, and the client-side Places package for Client applications or Server mux applications that log in on behalf of the users.

The Sametime Java API works in any development environment that is compatible with JDK 1.1 or greater. In the next few sections, we consider some of the more popular IDEs, and the Sametime-specific setup involved. For the purposes of this chapter, we placed the Sametime Client Toolkit binaries in the following directory:

```
C:\Sametime31\bin
```

We placed the Sametime Server Toolkit binaries in the following directory:

```
C:\Sametime31 CommServer Toolkit\bin
```

2.3.1 IBM WebSphere Studio Application Developer 5.0

The first thing to do to set up WebSphere Studio Application Developer for Sametime application development is to configure WebSphere Studio Application Developer's Java classpath. In order for WebSphere Studio Application Developer to find the required Sametime classes to compile and run your Java programs, you must add the toolkit binaries to WebSphere Studio Application Developer's classpath. Do the following steps:

1. Select **Window -> Preferences**.
2. Select **Java -> Classpath Variables**, then click the **New** button (see Figure 2-2).

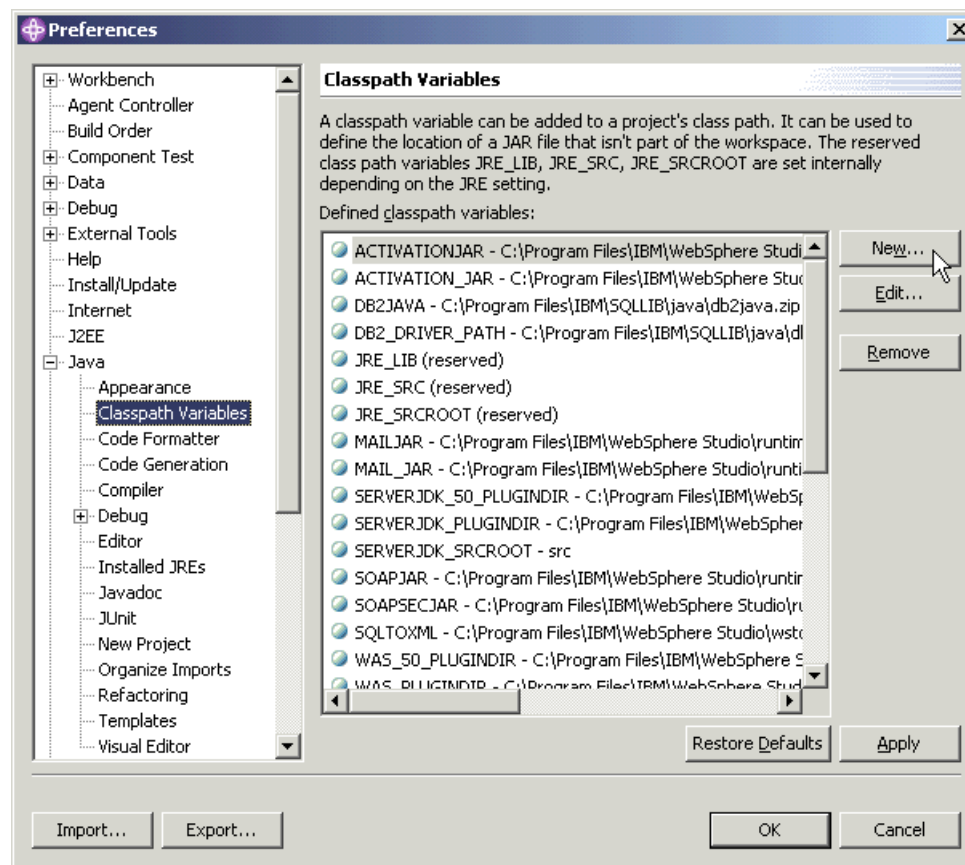


Figure 2-2 WebSphere Studio Application Developer's classpath variables

3. In the New Variable Entry, give the variable an appropriate name, enter the full path to the jar file or directory, and click **OK** (see Figure 2-3 on page 27).

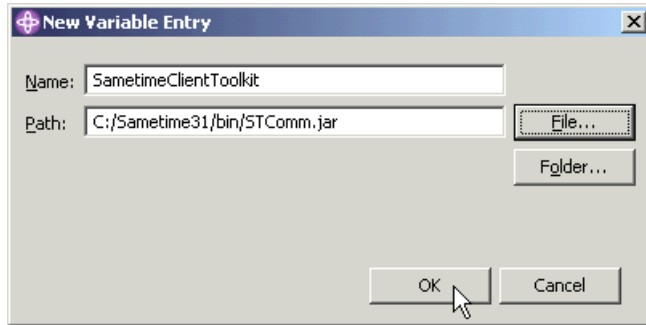


Figure 2-3 Adding a new classpath variable

4. Verify that the new variable appears in the list of classpath variables (see Figure 2-4 on page 28).

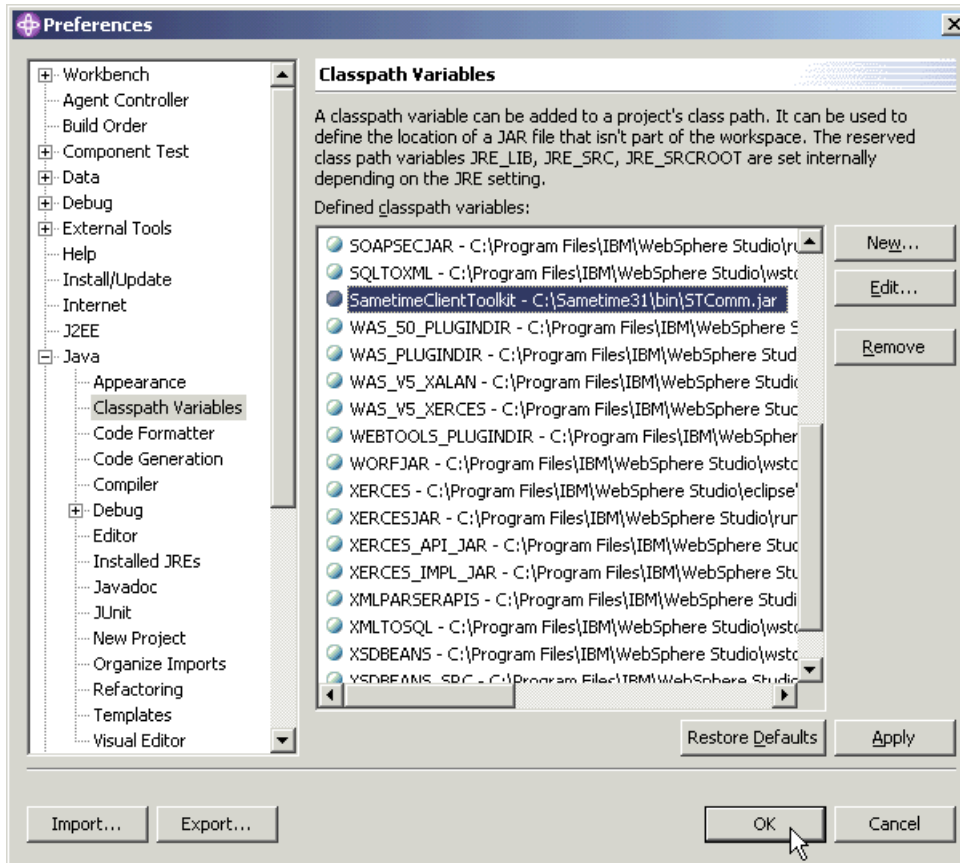


Figure 2-4 New entry for Sametime Client Toolkit variable

Tip: Depending on what you are developing, you may also have to add variables for some of the other jar files that are included with the Client and Server Toolkits. For example, if you are developing a server application, you will need to add a variable that points to the full path of the stcommsrvrtk.jar file.

5. Create a Java project to hold your Sametime application by selecting **File -> New -> Project**.
6. Select **Java** in the left list box and then **Java Project** in the right list box. Click **Next**.
7. Give the project a name, then click **Next** (see Figure 2-5 on page 29).

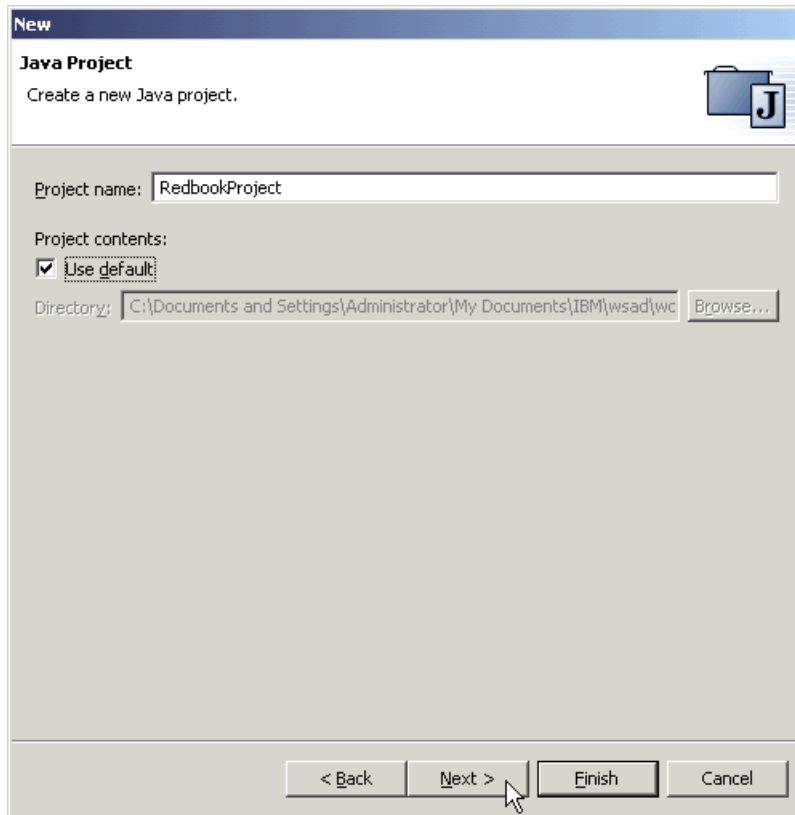


Figure 2-5 Java project details

8. On the Java Settings page, click the **Libraries** tab. You will see one default entry pointing to the JRE supplied by WebSphere Studio Application Developer. You can replace this with another one of your choosing, but we kept the default. Click the **Add Variable** button (see Figure 2-6 on page 30).

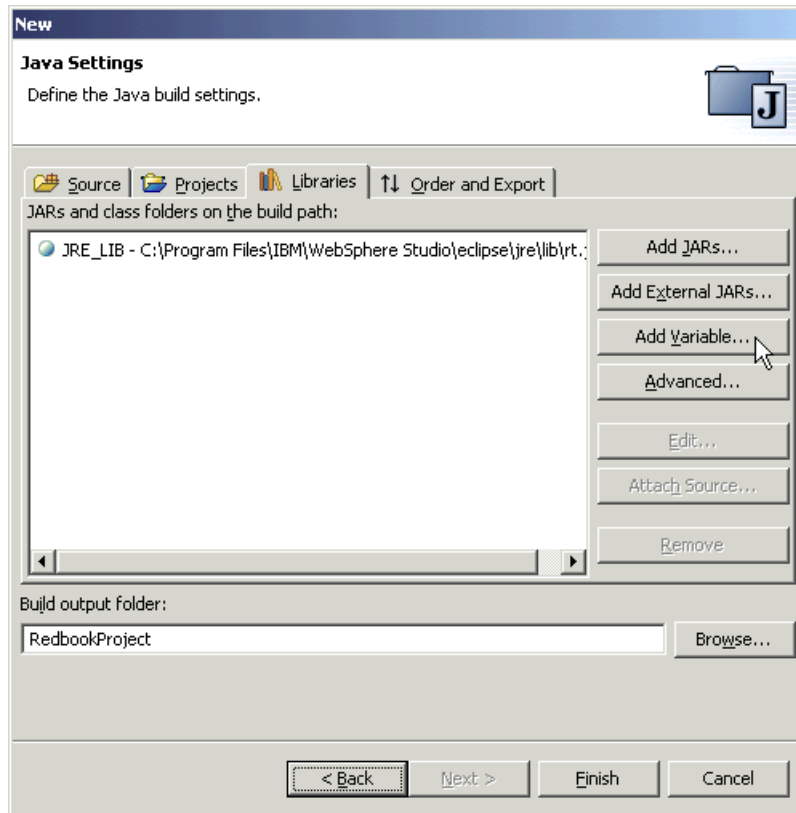


Figure 2-6 Adding a classpath variable to the project

9. Select the entry for the Sametime Client Toolkit jar file that you created in step 3, then click **OK** (see Figure 2-7).

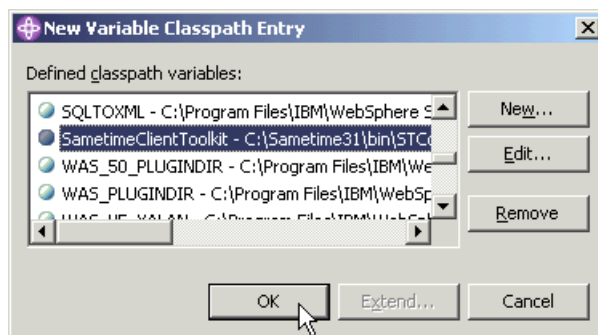


Figure 2-7 Select Client Toolkit variable

10. Click **Finish** to create the project. The newly created project will show in the left-hand view in WebSphere Studio Application Developer. If you expand the project, you will see the Sametime Client Toolkit jar file is specified as belonging to the project (Figure 2-8).

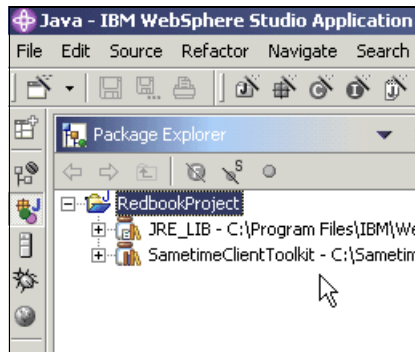


Figure 2-8 Sametime Client Toolkit included in project

11. You can now start writing your Java applications, or import a program from the file system.

2.3.2 Borland JBuilder

The steps involved in setting up JBuilder for Sametime development are very similar to those for WebSphere Studio Application Developer. You must add the toolkit binaries to JBuilder's classpath so it can compile and run your programs. Do the following steps:

1. Select **Tools -> Configure Libraries**.
2. Click the **New** button.
3. In the New Library Wizard dialog, give the library a name. Change the Location to be **JBuilder**, then click the **Add** button to add the toolkit binaries (see Figure 2-9 on page 32).



Figure 2-9 Adding a new library

4. Navigate to the location of the toolkit binaries, then click **OK** (see Figure 2-10 on page 33).

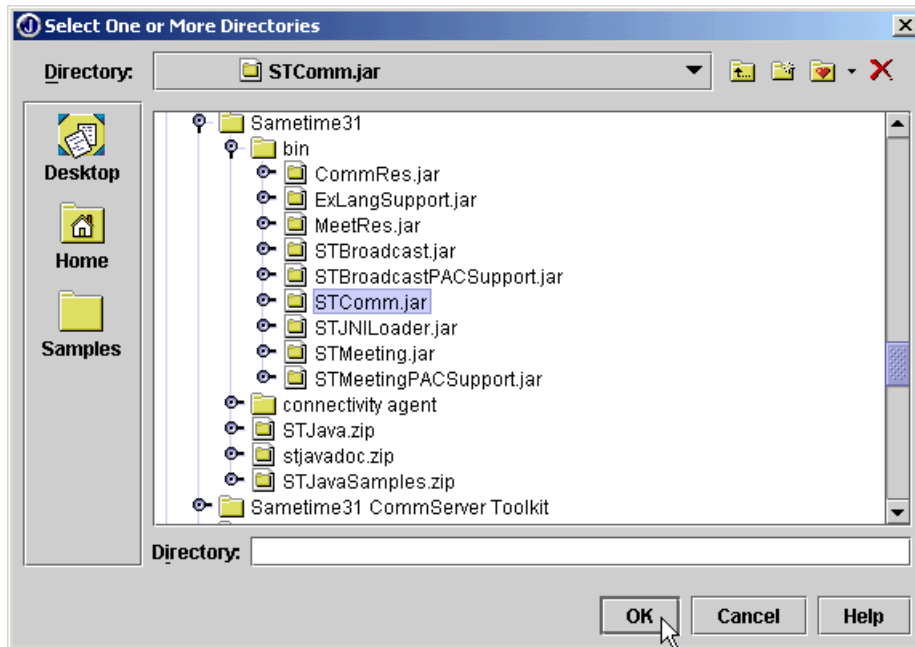


Figure 2-10 Locate the Client Toolkit binaries

5. Click **OK** when you have finished adding the binaries to your library.

Tip: Depending on what you are developing, you may also have to add libraries for some of the other jar files that are included with the Client and Server Toolkits. For example, if you are developing a server application, you will need to add a library that points to the full path of the stcommsrvrtk.jar file.

6. Create a new project to hold your Sametime application by selecting **File -> New Project**.
7. Give the project a name, then click **Next**.
8. On the next dialog page, click the **Required Libraries** tab, then click the **Add** button (see Figure 2-11 on page 34).

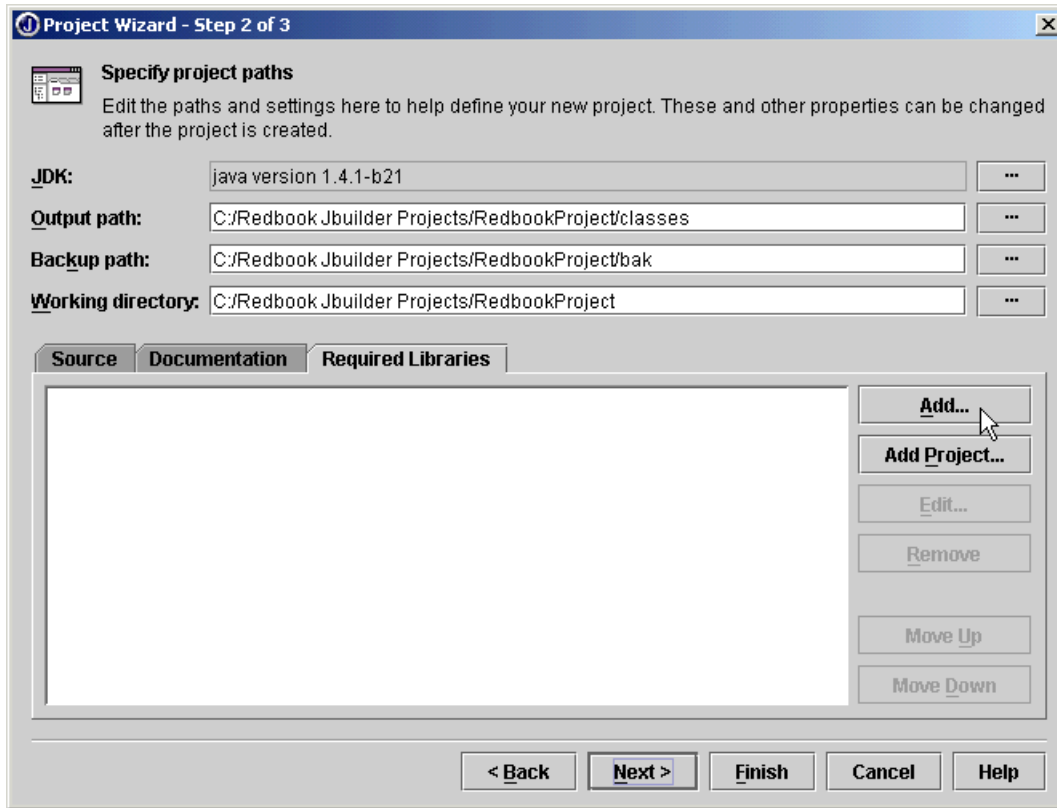


Figure 2-11 Adding a library to the project

9. Locate and highlight the library you created in step 3, then click **OK** (see Figure 2-12 on page 35).

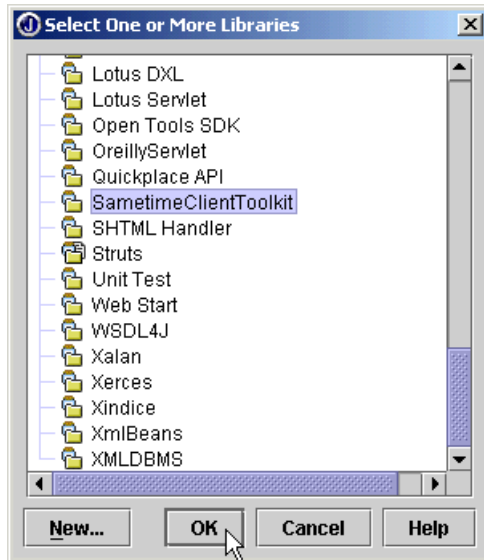


Figure 2-12 Select the Client Toolkit library from the list

10. Click **Next** then **Finish** to finish creating your project. You can now start writing your Java applications, or import a program from the file system.

2.3.3 Sun JDK

As mentioned earlier, the Sametime Java API supports any JDK Version 1.1 or higher. In order for Windows to find the required Sametime classes to compile and run your Java programs, you must add the toolkit binaries to the classpath variable.

To set up the classpath variable on Windows 2000, do the following:

1. Go to the **Start** menu.
2. Choose **Settings -> Control Panel**.
3. Double-click on the **System** icon.
4. Choose the **Advanced** tab in the System dialog window.
5. Click the **Environment Variables** button (see Figure 2-13 on page 36).

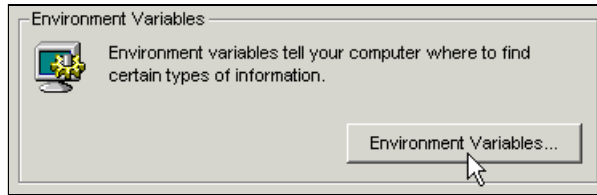


Figure 2-13 Environment Variables section of Advanced tab

6. In the **System Variables** section, find the variable called **Classpath**. Select it and click **Edit** (see Figure 2-14).

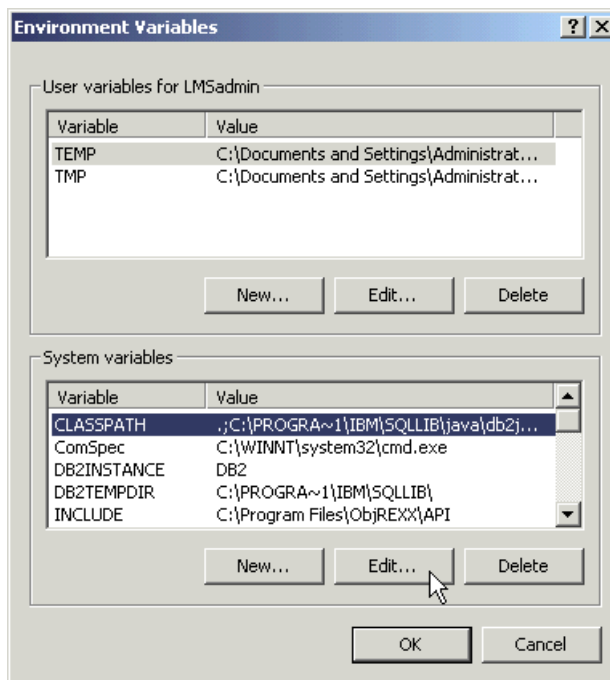


Figure 2-14 Environment Variables dialog box

7. In the variable value section, add the full path of the Sametime Toolkit jar files. For example, if the Client jar files are located in the C:\Sametime31\bin directory, then your classpath variable should look like Figure 2-15 on page 37.

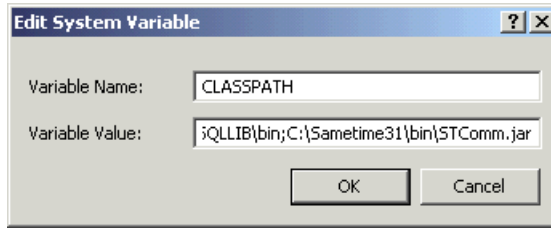


Figure 2-15 Editing the classpath variable

Tip: Depending on what you are developing, you may also have to add some of the other jar files that are included with the Client and Server Toolkits. For example, if you are developing a server application, you will need to add the full path of the stcommsrvrtk.jar file to your classpath.

8. Click **OK** to save your new classpath. Click the next **OK** button to close the dialog for environment variables. Finally, click **OK** to close the System dialog window.
9. You can now compile and run your Sametime programs from a command line.

2.3.4 Installing the C++ Toolkits

If installed on the Sametime server, all the Sametime Toolkits can be downloaded from the toolkits page.

1. Navigate to the home page of the Sametime server (http://<your_Sametime_server>/stcenter.nsf).
2. Click on the **SDK** link at the bottom of the page.
3. Click **C++ Toolkit** on the SDK page. You are now at the Sametime C++ Toolkit home page, as shown in Figure 2-16 on page 38.

Welcome to the Sametime 3.1 C++ Toolkit

The Sametime C++ Toolkit is a collection of static library components that allow developers to enhance Windows applications with Community Services, such as awareness and instant messaging. Developers can quickly and easily Sametime-enable any Win32 or MFC based Windows application.

The Toolkit supports the Microsoft® Visual C++ 6.0 Developer Studio SP5.

Getting Started

[Tutorial](#) (~2,2 MB pdf)

[Samples](#)

[Release Notes](#)

Additional Documentation

Figure 2-16 The Sametime C++ Toolkit home page

4. The Sametime C++ Toolkit is provided as a ZIP file. Click on **Toolkit, Binaries, and Header Files** to download the ZIP file.
5. Using a tool like PKZip or WinZip, extract the C++ Toolkit files to a directory on your hard disk, being sure to maintain the folder/directory structure. Figure 2-3 outlines the extracted directory structure.

Table 2-3 C++ Toolkit directory structure

Name of subfolder	Description
connectivity agent	Contains the StConnAgent31.exe. This executable is part of the connectivity agent framework.
inc	A set of subfolders that contain the C++ Header Files.
lib	A set of folders that contain the C++ Toolkit static libraries, including both the components implementing the different services and additional core libraries required for any Sametime-enabled application. The libraries are provided both in release and debug binary versions.

Note: For full details on the contents of each subfolder and for more details regarding the connectivity agent framework, consult the *Sametime 3.1 C++ Developers Guide* (see 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22 for information on finding this guide).

2.3.5 Microsoft Visual C++

Now that we have extracted the C++ Toolkit files, we need to include them in our Visual C++ Project. We need to configure support for any necessary include files and any static libraries we require.

Assuming that you have your C++ project open, configuring the include files involves a few simple steps:

1. Select **Tools** → **Options**, which will then display the options seen in Figure 2-17.

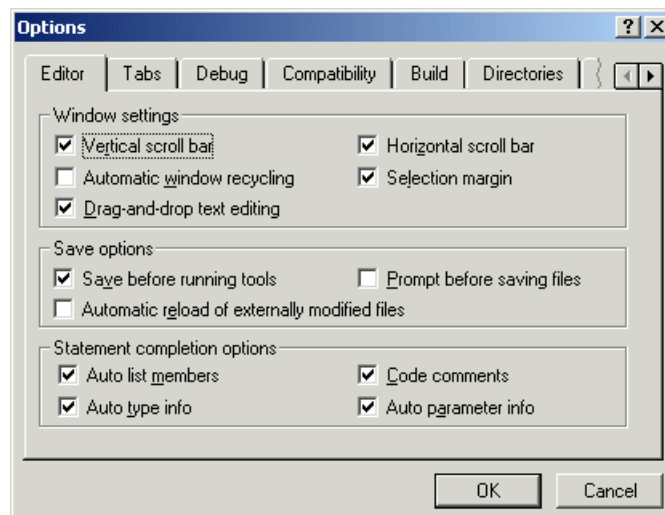


Figure 2-17 Microsoft Visual C++ options

2. Click on the tab labeled **Directories**, which will display the dialog seen in Figure 2-18 on page 40.

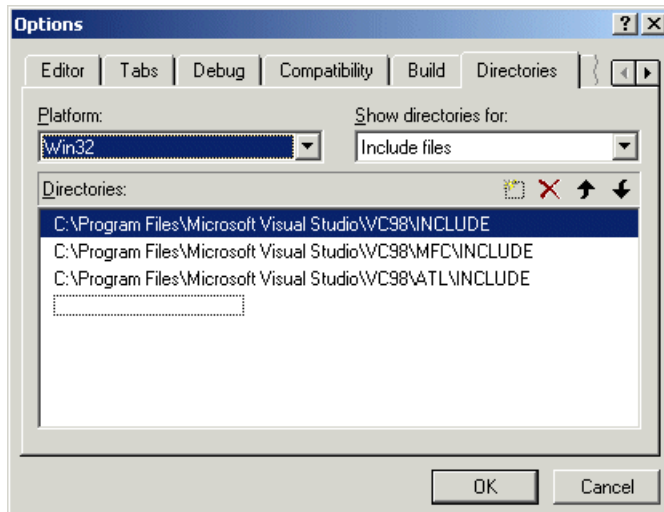


Figure 2-18 Directories options

3. From this screen, we want to add a new Directory for a set of include files, which is done by clicking on the **New** icon, as indicated in Figure 2-19.

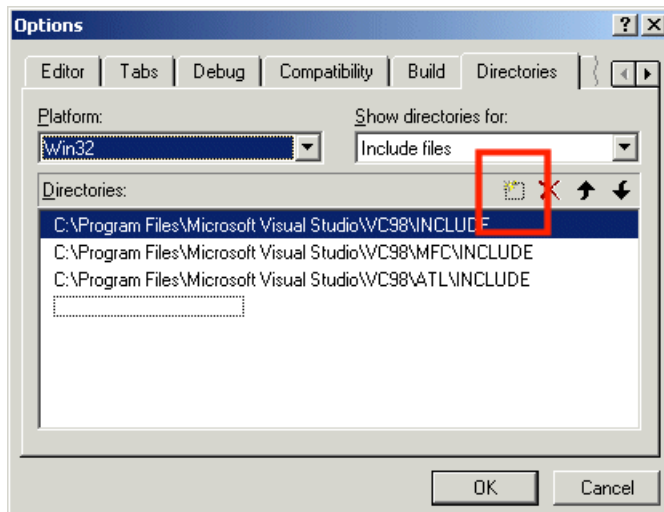


Figure 2-19 New directory icon

4. When the **New** icon is clicked, an additional line is added for the include directories. The directory that should be pointed to is the directory that has the include files you require. For some applications, you may only need a single directory; for others you may need many. This is the dialog where you

would add the additional directories that you require. For the purpose of this redbook, we will only include a single directory, the stcommon directory, as seen in Figure 2-20.

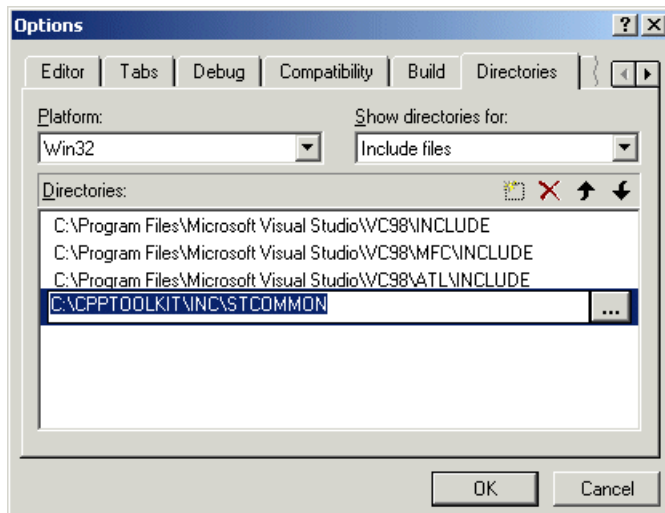


Figure 2-20 New include directory

5. To add the static libraries, we need to follow basically the same steps as 1-5, but after Step 2, we need to select **Library Files** in the **Show Directories** drop-down, as shown in Figure 2-21.

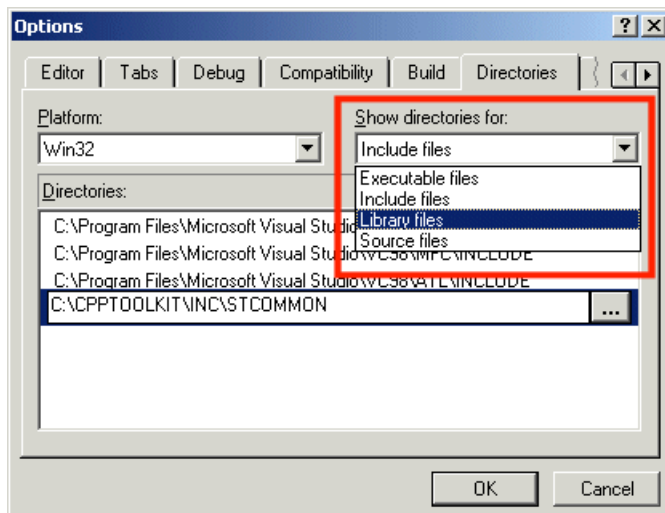


Figure 2-21 Selecting library files

6. Having selected the Library files option, we will now include the Static library into our project by clicking the **New** icon.
7. Entering the directory where the static libraries are located, choosing either the release libraries or the debug libraries. In Figure 2-22, we can see the settings for the release static libraries

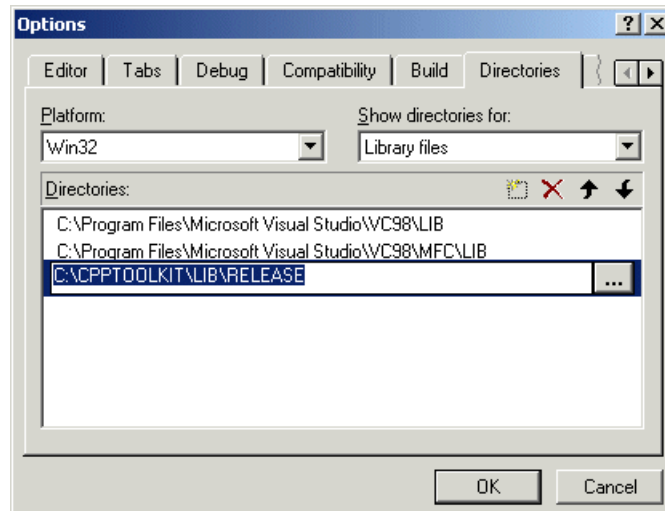


Figure 2-22 Including release static libraries

8. To make these changes effective, click on the **OK** button.

With these changes made, we can now use the C++ Toolkit in our project within Microsoft Visual C++.

2.4 Installing the COM Toolkit

We will now look at how to get and install the COM Toolkit, and then how you would add a reference to the toolkit in Visual Basic.

If installed on the Sametime server, all the Sametime Toolkits can be downloaded from the toolkits page.

1. Navigate to the home page of the Sametime server (http://<your_Sametime_server>/stcenter.nsf).
2. Click on the **SDK** link at the bottom of the page.

Click **COM Toolkit** on the SDK page. You are now at the Sametime COM Toolkit home page, as shown in Figure 2-23 on page 43.

Welcome to the Sametime 3.1 COM Toolkit

The Sametime COM Toolkit provides a COM object with basic Sametime Community Services for Visual Developers. This COM object provides login, awareness, and instant messaging – allowing the developer to create the required UI.

The Toolkit supports the Microsoft® Visual Basic 6.0 and Microsoft VBA of Windows 2000. It should also work in other environments that support COM and OLE automation.

Getting Started

[Developers Guide](#) (~420 KB pdf)

[Samples](#)

[Release Notes](#)

Additional Documentation

"Working With the Sametime Client Toolkits," [IBM Redbook](#),
IBM Form Number SG24-6666-00

[White Paper - "Introducing the Sametime Client Toolkits"](#) (~1.8 MB pdf)

Downloads

[Toolkit & Samples](#) (~2.0 MB self-extracting zip file)

Figure 2-23 The Sametime COM Toolkit home page

You can now download and install the COM Toolkit by clicking the **Toolkit & Samples** link on the COM Toolkit home page.

2.4.1 Microsoft Visual Basic

You can use the COM Toolkit in two ways: early binding and late binding. When you use the early binding method, you will need to add a library reference to your project. To add the COM Toolkit reference to your project, select **Project -> References**, as illustrated in Figure 2-24 on page 44.

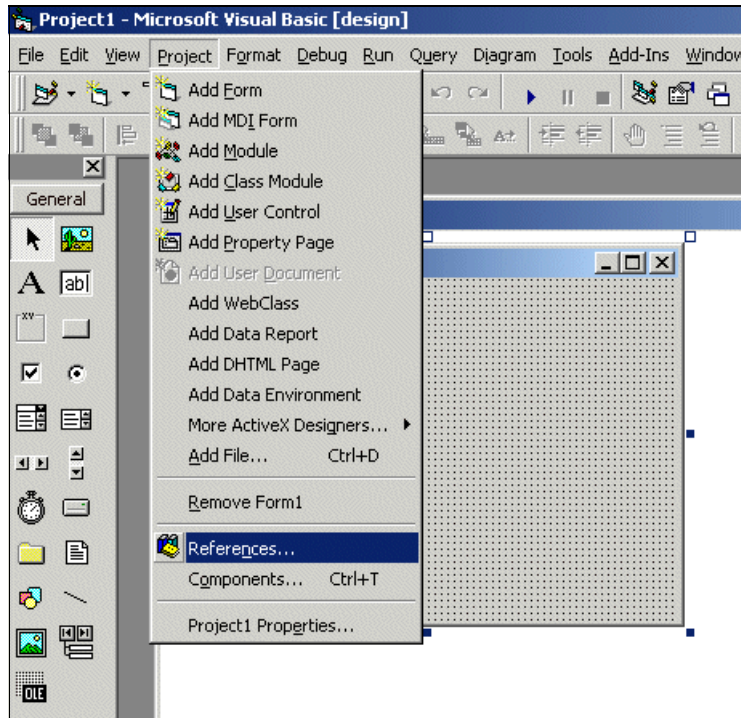


Figure 2-24 Selecting the References menu

After clicking on the **References** menu item, the References dialog is opened. Once the dialog is opened, search for STComTk.dll and select it. Then click **OK**, as shown in Figure 2-25 on page 45.

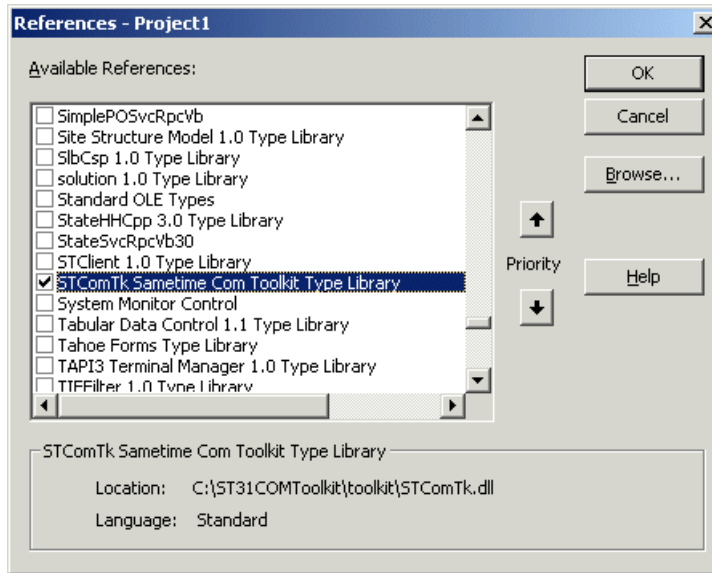


Figure 2-25 Selecting the COM Toolkit in the References Dialog

You can also use the late binding method. To use this method, you will need the STCOMTkLib library.



Part 2

Sametime enabling applications

Part 2 focuses on the techniques and methods for custom development with Sametime.



Sametime Bots

This chapter discusses the concept of Sametime Bots, what they are, and how to build them.

We start with a general discussion of what a bot is, before continuing to talk about what makes a Sametime Bot special.

We discuss the various ways that you can build a Sametime Bot using the available toolkits. We examine, in detail, a couple of bots developed using the Java Client Toolkit to show how they work and how you can create your own.

We conclude this chapter by suggesting enhancements to the bot concept, a discussion around possible deployment scenarios, and a whole host of ideas for potential bots.

For the purposes of this chapter, we assume you are familiar with the developer toolkits that come with Sametime; if you are not, you may wish to read the redbook *Working with the Sametime Client Toolkits*, SG24-6666.

3.1 What is a bot?

The term bot is derived from the word *robot*, and is used to describe an application that can take instructions from an end user and provide a suitable response.

A Sametime Bot is a powerful extension of this idea, incorporating the functionality of a normal bot application with the presence awareness and instant messaging capability of Sametime. Bots extend the capability of the Sametime instant messaging client to allow users to connect with applications as well as people. They provide an intuitive, interactive, and text-based interface to back-end information sources.

A Sametime Bot is a member of the Sametime community, and therefore has a presence that the rest of the community is aware of. A Sametime Bot appears in a user's buddylist, just like a regular Sametime user. This means that when the bot is running, it will appear online as just another member of the community. Figure 3-1 shows a Sametime Connect client with a sample buddylist containing a number of bots used internally by IBM.

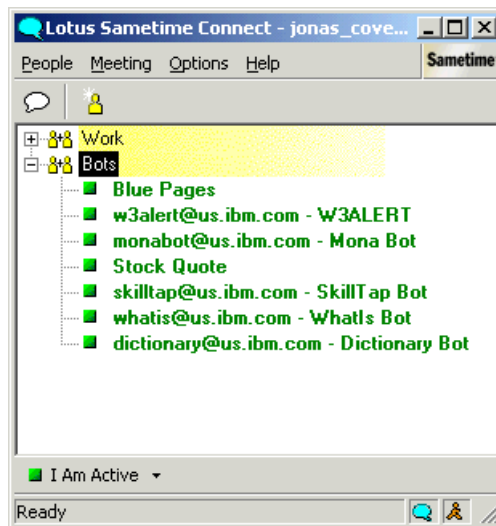


Figure 3-1 Sample buddylist showing Sametime Bots

This presence allows you to create bots that users can actually interact with. A bot can respond to a request for an instant chat, just like a regular Sametime user. In this way, a Sametime Bot can represent a virtual person, resembling an ever-present expert who can answer questions and execute commands instantly.

As long as it is logged in, a bot is continually available to service requests, and will respond as long as the Sametime server itself is available. In a multi-server community, the bot is available to any user, regardless of which Sametime server they are actually connected to. It is also worth noting that because a Sametime Bot manifests itself as just another member of the community, bots are available to mobile users with no extra programming effort. An end user can interact with a bot using a cell phone or PDA in the same manner he would communicate with a regular Sametime user.

3.2 Developing bots

The very nature of their client interaction means that most bots are developed using the Sametime Client Toolkits. These toolkits expose the Community Services that allow an application to log in to the Sametime community and to communicate with the end user via instant messages. This means you can use the following toolkits to build your own bots:

- ▶ Java
- ▶ C++
- ▶ COM

Note: It is possible to develop bots using the Sametime Community Server Toolkit, but the process is not so straightforward. Using the Server Toolkit is discussed in 3.4, “Enhancing the bot framework” on page 79.

Bots are written as applications that log in to the Sametime community as a specific user; for this reason, they require a corresponding entry in whichever directory the Sametime server uses for authentication, whether it be a Domino Directory or an LDAP server. For each bot that you develop, you should create a corresponding entry in the user directory, and supply the user name and password to the bot so it can log in. The server will use these details to authenticate the bot against the user directory.

In addition to the different toolkits that can be used to develop bots, there are a number of ways that each toolkit can be used. For example, if we consider using the Java Client Toolkit, we can create a bot in any of the following ways:

- ▶ As a stand-alone Java application that runs from the command line.
- ▶ As a Java servlet hosted on a Java application server, such as IBM WebSphere Application Server.
- ▶ As a Java applet contained within a standard Web page.

The method you choose to employ depends on your environment; there is no single pattern that all Sametime Bots must conform to. bots can run on either the client or the server. You may find it easier and more convenient to write and deploy your bots as servlets, for example, if you have access to a suitable application server.

Regardless of which approach you take to create your bot, there are a few fundamental things that all bots must do:

1. Create a Sametime session and load the necessary components.
2. Log in to the Sametime community.
3. Register the type of messages the bot should receive.
4. Listen for any incoming instant messages.
5. Respond accordingly; this is where the bot-specific application logic should be inserted, to analyze the user's request and select an appropriate response or action.

Using the Java Client Toolkit, we will now look at these five steps in greater detail.

3.2.1 Creating a Sametime session

The first thing any bot application must do is create a Sametime session. Every Sametime session must have a unique name that can be used as a key to access it from the static session table that the server uses to hold all session objects. We can use the Java keyword `this` to help derive a unique name. If the session name is already in use, an exception will be thrown that must be caught.

Example 3-1 contains an example of creating an `STSession` object.

Example 3-1 Creating an STSession object

```
try
{
    m_session = new STSession("MySametimeBot" + this);
}
catch (DuplicateObjectException doe)
{
    doe.printStackTrace();
    return;
}
```

If the Sametime session is created successfully, the next step is to load the session into the Sametime component functions needed by the bot. Sametime functionality is split among several components, each responsible for providing a

service. The Sametime Toolkits include components for accessing the individual services provided by the Sametime server. The STSession object holds the set of components associated with the Sametime session.

There are three methods in the STSession class that can be used to load components into the session:

- ▶ `loadAllComponents()`: Loads all components, both UI and semantic (non-UI).
- ▶ `loadComponents()`: Loads a list of components according to their class names.
- ▶ `loadSemanticComponents()`: Loads just the semantic components.

Sametime Bots do not require any UI components, so we use the `loadSemanticComponents()` method:

```
m_session.loadSemanticComponents();
```

Once the necessary components have been loaded, we start the session:

```
m_session.start();
```

3.2.2 Logging in to the community

Once the Sametime session has been started, the bot can attempt to log in to the Sametime community. In order to do this, we need to get a handle on the `CommunityService` component that was loaded into the session in the previous step. We do this with the `STSession.getCompApi()` method:

```
m_communityService =  
(CommunityService)m_session.getCompApi(CommunityService.COMP_NAME);
```

We must cast the resulting object into an instance of `CommunityService`, as the `STSession` object is unable to differentiate between the various components.

We then encounter the first example of the Sametime event-driven programming model. Most Sametime development involves registering as a listener for a specific event, then responding accordingly when that method is called by the server. Logging in to the community is no different; the bot registers itself as a listener to the community login event by implementing the `LoginListener` interface and the `addLoginListener()` method of the `CommunityService` object:

```
m_communityService.addLoginListener(this);
```

Now that the bot is registered as a listener, the Community Service will notify it when the login process has concluded. The final step in this process is to actually log in. This is done by calling one of the three login methods of the `CommunityService` object:

- ▶ `loginByPassword(servername, username, password);`

- ▶ `loginByToken(servername, username, token);`
- ▶ `loginAsAnon(servername, username);`

For example, to log in using the password of the bot as stored in the user directory, you would use this method:

```
m_communityService.loginByPassword(servername, mybotname, mybotpassword);
```

Upon completion of the login process, the Community Service will call one of two methods: `loggedIn()` or `loggedOut()`. Both of these methods are defined in the `LoginListener()` interface and so must be implemented by the bot.

A successful login to the Sametime server will result in the `loggedIn()` method being called. Conversely, if the login attempt failed or was refused, the `loggedOut()` method is called (you can use the `LoginEvent` object that is passed in to this method to get a reason code as to why the login failed):

```
public void loggedOut(LoginEvent e)
{
    System.out.println("Logged Out, reason: " + e.getReason());
}
```

Tip: These error codes are explained in the JavaDoc for the `com.lotus.sametime.core.constants.STError` class.

3.2.3 Registering the message type

Once the bot has successfully logged in to the Sametime community, we start to invoke the functionality necessary for it to behave like a virtual person. For the bot to communicate with regular Sametime users, it must use the `InstantMessagingService` component and register to receive chat messages. We need to get a handle on the `InstantMessagingService` component from the session object. We then register that we are interested in receiving messages of type `IM_TYPE_CHAT`; we can do this with the `loggedIn()` method. It is very important that we register the type of message we are interested in, as the `InstantMessagingService` will reject any message that does not have a registered IM type. This task is done as follows:

```
m_imService =
    (InstantMessagingService)m_session.getCompApi(InstantMessagingService.COMP_NAME
    );
m_imService.registerImType(ImTypes.IM_TYPE_CHAT);
```


It is important for our bot to know when a message arrives from an end user. To accomplish this, the bot implements the `ImServiceListener` interface and registers to listen for events using the `addImServiceListener()` method of the `InstantMessagingService`:

```
m_imService.addImServiceListener(this);
```

The `ImServiceListener` interface defines the `imReceived()` method that must be implemented by the bot. This method is called when a user initiates a conversation with the bot. It is triggered as soon as the user double-clicks on the bot's name in her buddylist. The bot can use this event to send a greeting message to the user, perhaps with instructions on how to use the bot effectively or personalized information specific to the user.

Important: When a user clicks on a Sametime Link to launch a chat window, the `imReceived()` method is not called. For a discussion on using bots with Sametime Links, see 9.6, “Using Sametime Links with bots” on page 256.

3.2.4 Listening for incoming messages

Once the `imReceived()` method is triggered, the bot must implement the `ImListener` interface to receive follow-on messages from the user. It uses the `addImListener()` method of the `Im` object to add itself as an `ImListener`. The bot gets a handle on the `Im` object from the `ImEvent` object that is passed in to the `imReceived()` method:

```
public void imReceived(ImEvent e)
{
    e.getIm().addImListener(this);
}
```

The `ImListener` interface defines five methods, one of which, `textReceived()`, is called when the user sends a text message to the bot. The actual content of the text message can be retrieved by calling the `getText()` method on the `ImEvent` object that is passed in to the `textReceived()` method:

```
public void textReceived(ImEvent e)
{
    String messageFromUser = e.getText();
}
```

3.2.5 Responding with logic

Once the bot has received a text message from the end user, it needs to perform any bot-specific logic and respond back. For example, consider a Dictionary Bot that a user sends words to and expects back their meaning. The `getText()`

method returns the word the end user is searching for. The bot then calls specific functions to determine the meaning of the word, and sends that meaning back to the end user with the `sendText()` method of the `Im` object:

```
public void textReceived(ImEvent e)
{
    String dictionaryWord = e.getText();
    String wordMeaning = botDictionaryLogic(dictionaryWord);
    e.getIm().sendText(true, wordMeaning);
}
```

Tip: The first parameter of the `sendText()` method is a boolean flag that determines whether or not the text should be sent encrypted.

3.3 Bot examples

This section details three bots developed using the Java Client Toolkit. The first, the Echo Bot, is a Java application that simply echoes back any text that is sent to it. The second is an FAQ Bot, developed as a Java servlet that reads and writes to a Domino database containing the FAQs. The third is a Translation Bot that uses a Web service to convert phrases entered by the user into the language of their choice. All of these bots follow the five essential steps detailed in the previous section.

3.3.1 The Echo Bot

This first example shows the basic framework of how to construct a bot, incorporating all the stages outlined in the previous section. The Java application `EchoBot` takes three command line parameters:

- ▶ The Sametime server DNS name or IP address
- ▶ The user name of the bot as stored in the Sametime directory
- ▶ The password for that user

Example 3-2 shows the `EchoBot.java` code.

Example 3-2 EchoBot.java

```
import com.lotus.sametime.community.*;
import com.lotus.sametime.core.comparch.*;
import com.lotus.sametime.core.constants.*;
import com.lotus.sametime.im.*;

public class EchoBot
    implements Runnable, LoginListener, ImServiceListener, ImListener
{
```

```

private Thread botThread;
private STSession m_session;
private CommunityService m_communityService;
private InstantMessagingService m_imService;

public EchoBot(String serverName, String botName, String botPassword)
{
    try
    {
        m_session = new STSession("EchoBot " + this);
    }
    catch (DuplicateObjectException doe)
    {
        doe.printStackTrace();
        return;
    }

    m_session.loadSemanticComponents();
    m_session.start();

    m_communityService =
(CommunityService)m_session.getCompApi(CommunityService.COMP_NAME);
    m_communityService.addLoginListener(this);
    m_communityService.loginByPassword(serverName, botName, botPassword);

}

public void loggedIn(LoginEvent e)
{
    System.out.println("Logged In");

    m_imService =
(InstantMessagingService)m_session.getCompApi(InstantMessagingService.COMP_NAME
);
    m_imService.registerImType(ImTypes.IM_TYPE_CHAT);
    m_imService.addImServiceListener(this);

}

public void loggedOut(LoginEvent e)
{
    System.out.println("Logged Out");
}

public void imReceived(ImEvent e)
{
    e.getIm().addImListener(this);
    System.out.println("IM Received");
    // You could send back a greeting at this point

```

```

        // or instructions on how to use the bot...
        e.getIm().sendText(true, "Welcome to the Echo Bot...");
    }

    public void dataReceived(ImEvent e) {}

    public void imClosed(ImEvent e) {}

    public void imOpened(ImEvent e) {}

    public void openImFailed(ImEvent e) {}

    public void textReceived(ImEvent e)
    {
        String messageText = e.getText();
        System.out.println("Message received from " +
            e.getIm().getPartner().getName());

        // This is the point to insert any application-specific code...
        // Here we just echo back what the end user sent...
        e.getIm().sendText(true, "You typed: " + messageText);
    }

    public void start()
    {
        if (botThread == null)
        {
            botThread = new Thread(this, "EchoBotThread");
            botThread.start();
        }
    }

    public void run()
    {
        Thread myThread = Thread.currentThread();
        while (botThread == myThread)
        {
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {}
        }
    }

    public static void main(String[] args)
    {
        if (args.length != 3) {
            System.out.println("Usage: EchoBot serverName botName botPassword");
        }
    }

```

```
        System.exit(0);
    }

    EchoBot echoBot = new EchoBot(args[0], args[1], args[2]);
    echoBot.start();
}

}
```

Running the application

The bot can be run from a command prompt. Figure 3-2 shows how the Echo Bot gets the server name, user name, and password from the command line and uses them to log in to Sametime when the application starts.

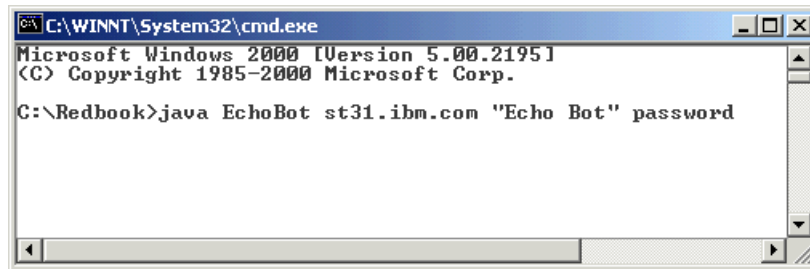


Figure 3-2 Running the Echo Bot from a command line

If the bot successfully logs in to the Sametime server, Figure 3-3 on page 60 shows a user's buddylist with the bot added to it.

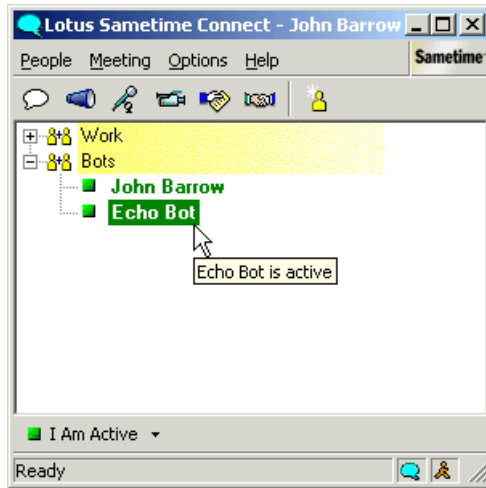


Figure 3-3 Buddylist showing bot online

If the user then starts a chat with the bot by double-clicking on its name, the `imReceived()` method is called and, in this example, the bot responds by sending a greeting message, as shown in Figure 3-4.

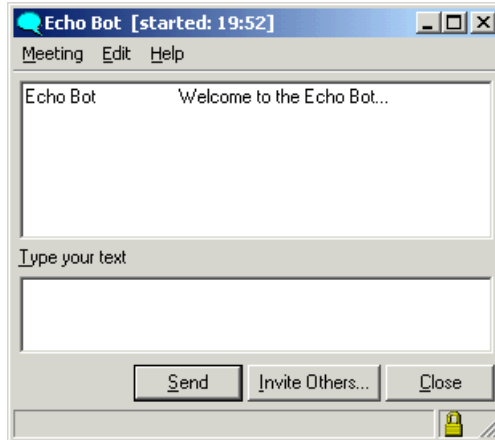


Figure 3-4 Bot sends greeting in response to `imReceived()` event

When the end user sends a message to the bot, the `textReceived()` method is called. This is where any application-specific code should be implemented. For example, in a Directory Bot, you could do a lookup on a Domino Directory here using the Domino Java API, or search an LDAP directory, or both. As Figure 3-5 on page 61 shows, in this example, we simply echo back what the user sent.

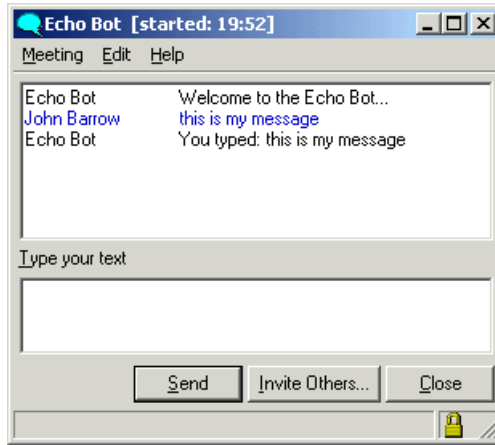


Figure 3-5 Bot responds with application-specific logic

3.3.2 FAQ Bot

This second bot builds upon the basics outlined in the Echo Bot example from the previous section. It is a Java servlet that accesses a back-end Domino database to retrieve and store information. It is made up of two classes:

- ▶ **SametimeBot:** An abstract class that can be used as a general bot framework class.
- ▶ **FAQBot:** Extends **SametimeBot** and implements the application-specific code.

SametimeBot

SametimeBot is an abstract class, that is, it cannot be instantiated itself, but it can be extended by application-specific classes such as **FAQBot**. It imports the basic Sametime classes a bot needs, as well as servlet-specific and Java utility packages. Being a servlet, it extends the **HttpServlet** class and implements the three ‘standard’ listeners a bot uses: **LoginListener**, **ImServiceListener**, and **ImListener**.

Example 3-3 shows the **SametimeBot** imports and class definitions.

Example 3-3 *SametimeBot* imports and class definition

```
import javax.servlet.*;
import javax.servlet.http.*;

import java.io.*;
import java.util.*;
```

```
import com.lotus.sametime.community.*;
import com.lotus.sametime.core.comparch.*;
import com.lotus.sametime.core.constants.*;
import com.lotus.sametime.im.*;

abstract class SametimeBot
    extends HttpServlet
    implements LoginListener,
        ImServiceListener,
        ImListener
```

The class then defines variables for the STSession, CommunityService, and InstantMessagingService objects, as well as four methods that are overridden in each bot subclass to return the welcome greeting, server name, user name, and password. A Vector object is created to keep track of the bot's open IM sessions (see Example 3-4 for details).

Example 3-4 Variable definitions

```
abstract protected String getWELCOME_TEXT();
abstract protected String getSERVER();
abstract protected String getAPP_NAME();
abstract protected String getAPP_PASSWORD();

private STSession m_session;
private CommunityService m_communityService;
private InstantMessagingService m_imService;

protected Vector m_imOpened = new Vector();
```

The init() method (Example 3-5) is called when the servlet is loaded into the servlet container. Here we attempt to log in to the Sametime server.

Example 3-5 init() method

```
public void init() throws ServletException
{
    System.out.println(this.getServletName() + ": init()");
    try
    {
        login();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```



```
}
```

In the `login()` method (Example 3-6), we use the `loginByPassword()` method to log in to the server.

Example 3-6 login() method

```
private void login()
{
    try
    {
        m_session = new STSession(this.getServletName());
        m_session.loadSemanticComponents();
        m_session.start();
    }
    catch (DuplicateObjectException doe)
    {
        doe.printStackTrace();
    }

    m_communityService = (CommunityService)
m_session.getCompApi(CommunityService.COMP_NAME);
    m_communityService.addLoginListener(this);
    m_communityService.loginByPassword(getSERVER(), getAPP_NAME(),
getAPP_PASSWORD());
}
```

In the `loggedIn()` method (Example 3-7) we register to receive chat messages and add the class as an `ImServiceListener`.

Example 3-7 loggedIn() method

```
public void loggedIn(LoginEvent event)
{
    System.out.println(this.getServletName() + ": loggedIn()");
    m_imService = (InstantMessagingService) m_session.getCompApi(
        InstantMessagingService.COMP_NAME);
    m_imService.registerImType(ImTypes.IM_TYPE_CHAT);
    m_imService.addImServiceListener(this);
}
```

In the `imReceived()` method (Example 3-8 on page 64), the bot checks the `m_imOpened` vector to see if it already has an open IM session with the user. If it does not, it adds the `Im` object to the vector, registers as an `ImListener`, and sends back the application-specific welcome text.

Example 3-8 imReceived method()

```
public void imReceived(ImEvent event)
{
    System.out.println("imReceived()");
    Im im = event.getIm();
    String name = im.getPartner().getDisplayName();

    boolean imExist = false;
    Im currentIm = null;
    for (int i = 0; i < m_imOpened.size(); i++)
    {
        currentIm = (Im) m_imOpened.elementAt(i);
        if (currentIm.getPartner().getId().equals(im.getPartner().getId()))
        {
            imExist = true;
            im = currentIm;
            break;
        }
    }
    if (!imExist)
    {
        m_imOpened.addElement(im);
        im.addListener(this);
        if (name.indexOf(" ") > 0)
        {
            name = name.substring(0, name.indexOf(" "));
        }
        String returnText = "Hello " + name + " - " + getWELCOME_TEXT();
        im.sendText(true, returnText);
    }
}
```

The `textReceived()` method (Example 3-9) is where the bot-specific application code is called. The `ImEvent` object passed into the method and the text sent by the user is passed into the abstract `doTextReceived()` method, which is overridden in each bot subclass. The result of any processing is sent back to the user (in this case, any matches to the question posed by the user).

Example 3-9 textReceived() method

```
public void textReceived(ImEvent event)
{
    Im im = event.getIm();
    String partnerName = im.getPartner().getDisplayName();
    String receivedText = event.getText().trim();

    // ignore no command
    if(receivedText.length() == 0)
```

```

        return;

        String returnText = "";
        returnText = doTextReceived(event, receivedText);

        im.sendText(true, returnText);
    }

    abstract protected String doTextReceived(ImEvent event,
                                              String strReceivedText);

```

If the end user closes their chat window, the `imClosed()` method (Example 3-10) is called. It makes sense at this point to remove them from the `m_imOpened` vector of open `Im` objects, close that particular `Im` object, and to remove the `ImListener`.

Example 3-10 imClosed() method

```

public void imClosed(ImEvent event)
{
    Im im = event.getIm();
    Im currentIm = null;
    for (int i = 0; i < m_imOpened.size(); i++)
    {
        currentIm = (Im) m_imOpened.elementAt(i);
        if (currentIm.getPartner().getId().equals(im.getPartner().getId()))
        {
            m_imOpened.removeElement(im);
            im.close(0);
            im.removeImListener(this);
            break;
        }
    }
}

public void imOpened(ImEvent event) {}

public void openImFailed(ImEvent event) {}

public void dataReceived(ImEvent event) {}

```

In order to keep the bot permanently active, in the `loggedOut()` method (Example 3-11 on page 66), we attempt to log the bot back in to the server. The `destroy()` method is also called when the servlet container unloads the servlet.

Example 3-11 loggedOut() method

```
public void loggedOut(LoginEvent event)
{
    if (m_session.isActive())
    {
        destroy();
    }
    while (!m_session.isActive())
    {
        try {
            Thread.sleep(5000);
            login();
        }
        catch (Exception e) {
        }
    }
}

/**Clean up resources*/
public void destroy()
{
    m_communityService.logout();
    m_communityService.removeLoginListener(this);
    m_imService.removeImServiceListener(this);
    m_session.stop();
    m_session.unloadSession(); }
}
```

FAQBot

This class is application-specific; it extends the SametimeBot abstract class and implements the logic specific to an FAQ Bot. In this case, it searches a Domino database for answers to questions posed by end users. It makes a note, by creating a new document, of any questions it does not already know the answer to, so the database can be expanded.

As our back-end data source is a Domino database and we will be accessing it using the Java API, we need to import the lotus.domino package (see Example 3-12).

Example 3-12 FAQBot package imports and class definition

```
import java.util.*;

import lotus.domino.*;

import com.lotus.sametime.im.*;

public class FAQBot
```

The class then overrides the four methods that are defined in the parent class with application-specific values (see Example 3-13).

Example 3-13 Class specific method overrides

```
protected String getWELCOME_TEXT()
{
    return "welcome to the FAQ Bot.";
}

protected String getSERVER()
{
    return "itsotest-st31.cam.itso.ibm.com";
}

protected String getAPP_NAME()
{
    return "FAQ Bot";
}

protected String getAPP_PASSWORD()
{
    return "mybotpassword";
}
```

We then come to the application-specific logic section of the bot. The SametimeBot class defines an abstract doTextReceived() method (Example 3-14). It takes an ImEvent object (from the textReceived() method) and the text the user sent as parameters. This method is then overridden in the FAQBot subclass and any other classes created that extend SametimeBot.

Example 3-14 doTextReceived() method

```
protected String doTextReceived(ImEvent event, String receivedText)
{
    Im im = event.getIm();
    String returnText = "";
    System.out.println(this.getServletName() + ": " +
        im.getPartner().getDisplayName() + " is requesting : " +
        receivedText);

    returnText = replaceSubString(receivedText, "?", "");
    returnText = replaceSubString(returnText, " ", "");

    return questionSearch(returnText.toUpperCase().trim());
}
```

```
}
```

In this particular application, the bot calls the `questionSearch()` method (Example 3-15) to perform the actual search of the back-end Notes database. It creates a Notes session (in this example, using the remote CORBA classes), then does a full-text search of the database for the phrase the end user entered. If it finds a match, it returns the answer text contained within the matching question document. If there is more than one match, the bot displays all matches and the user is prompted to refine their search. If there are no matches, a new document is created in the database so that this particular question may be added to the list of frequently asked ones if appropriate.

Example 3-15 `questionSearch()` method

```
public String questionSearch(String query)
{
    String strQuestionSearch = "";
    Session session = null;

    try
    {
        //Start a notes session to access the FAQBot config database
        NotesThread.sinitThread();
        session = NotesFactory.createSession("itsotest-st31.cam.itso.ibm.com");
        Database db = session.getDatabase("itsotest-st31/ITSOST31",
        "FAQBotConfig.nsf");

        if (db == null)
        {
            System.out.println("Can't get a handle on the Database");
        }
        else
        {
            //Do a full text search on the database
            DocumentCollection dc = db.FTSearch("[PossibleQuestions] Contains \"" +
            query + "\"");

            Document doc;

            if (dc.getCount() > 1)
            {
                //If there is more than 1 match...
                strQuestionSearch = "Do you mean: ";
                doc = dc.getFirstDocument();
                while (doc != null)
                {
                    strQuestionSearch = strQuestionSearch +
                    doc.getItemValueString("TrueQuestion");
                    doc = dc.getNextDocument(doc);
                }
            }
        }
    }
}
```

```

        if (doc != null)
        {
            strQuestionSearch = strQuestionSearch + " OR ";
        }
    }
}
else if (dc.getCount() <= 0)
{
    //If there are no matches...
    strQuestionSearch =
        "Sorry I don't understand you, type help for some possible
commands.";
    //Create a document in the Notes database
    doc = db.createDocument();
    doc.replaceItemValue("Form", "QuestionLog");
    doc.replaceItemValue("LogQuestionAsked", query);
    doc.save();
}
else
{
    //Return the answer
    doc = dc.getFirstDocument();
    strQuestionSearch = doc.getItemValueString("Answer");
}
}
}
catch (NotesException en)
{
    System.out.println(en.getMessage());
    en.printStackTrace();
}
catch (Exception e)
{
    System.out.println(e.getMessage());
}
finally
{
    try
    {
        if (session != null)
        {
            session.recycle();
        }
    }
    catch (Exception x)
    {
        x.printStackTrace();
    }
    NotesThread.stermThread();
}

```

```

    }

    return strQuestionSearch;
}

public String replaceSubString(String s, String findString,
                               String replaceString)
{
    StringTokenizer st = new StringTokenizer(s, findString, false);
    String t = "";
    do
    {
        if (!st.hasMoreElements())
        {
            break;
        }
        t = String.valueOf(t) + String.valueOf(st.nextElement());
        if (st.hasMoreElements())
        {
            t = String.valueOf(t) + String.valueOf(replaceString);
        }
    }
    while (true);
    return t;
}

```

Tip: If the servlet is located on a server that is not a Domino or Sametime server, you will be using the Java API to make remote calls, so you will need to make sure that the ncso.jar file is on your classpath.

The back-end Domino database is very simple: it contains a list of frequently-asked questions, along with their answers. Figure 3-6 on page 71 shows the view in the database that contains the question and answer documents.

Create New Question	
Question	Answer
Goodbye	Goodbye, thanks for using the FAQ Bot
Help?	I can do only a few things now but after someone te
How old are you?	I am not really any age, I was created just recently. I
Thanks	It is no problem at all.
What are you?	I am just a computer program
What is Sametime?	Sametime is an instant messaging technology, creat
What is your name?	As yet I haven't been given a name.
Where are you from?	I am not really from anywhere, to be honest I am just

Figure 3-6 View in Notes database showing FAQs and answers

Each question document contains a question and answer, and also a field for possible variations on the question. The ability to use this multi-valued field is a powerful feature of Domino's full-text search functionality. Figure 3-7 shows a sample question document.

FAQ Bot Config Question	
True Question:	What do you do?
Possible Questions:	WHAT ARE YOU WHAT DO YOU DO WHO ARE YOU
Answer:	I am just a computer program

Figure 3-7 Sample question document showing variations and answer

Once the two Java class files (SametimeBot and FAQBot) have been deployed to a servlet engine, the bot can be started by requesting the URL of the FAQBot servlet in a browser:

`http://yourserver/servlet/FAQBot`

This will call the `init()` method defined in the SametimeBot class, which will attempt to login the bot. Figure 3-8 on page 72 shows a buddylist with the FAQ Bot added to it.

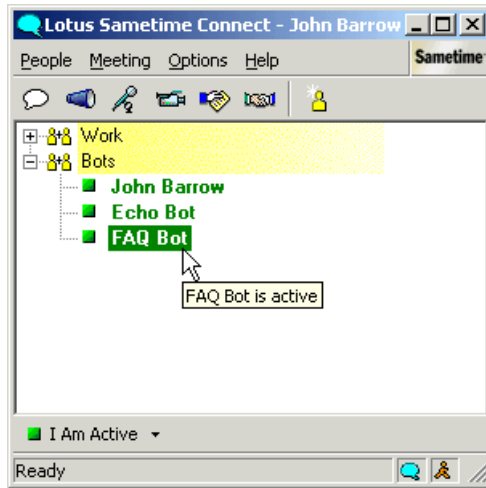


Figure 3-8 Buddylist with FAQ Bot online

If an end user initiates a chat with the bot, it responds with the welcome text defined in the `getWELCOME_TEXT()` method of the `FAQBot` class. Figure 3-9 shows the bot sending the welcome message.



Figure 3-9 Bot sends customized greeting in response to `imReceived()` event

When the end user enters a question, the bot calls the `doTextReceived()` method to carry out a search on the database and return an answer, if it finds one. Figure 3-10 on page 73 shows the bot returning an answer to a user's question.

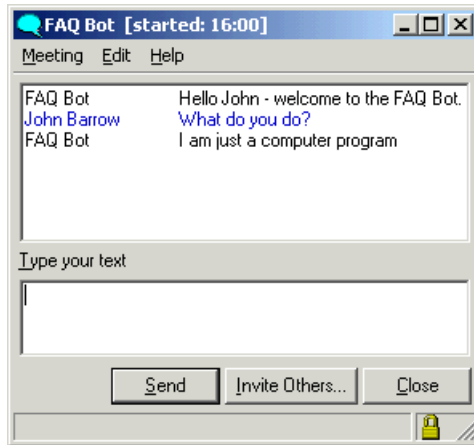


Figure 3-10 Bot responds with search results

If the bot cannot find the answer to the user's question, it responds back accordingly and creates a new document in the Domino database. Figure 3-11 shows the response sent back to the user.

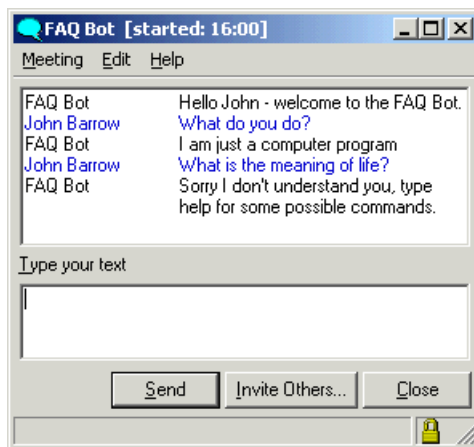


Figure 3-11 Message returned when no matching answer found

Figure 3-12 on page 74 shows the new Notes document created by the bot.


 Create New Question	
Question Asked	
	WHAT IS THE MEANING OF LIFE

Figure 3-12 Document created by bot for possible inclusion in FAQs

3.3.3 Translator Bot

This third bot is also built upon the SametimeBot abstract class introduced in “SametimeBot” on page 61. It uses the BabelFish Web service hosted by XMethods (<http://www.xmethods.net>) to provide language translation. The end user enters a two digit code representing their own language, and a two digit code representing the language they wish to receive the translation in. The user then types in the word or phrase they wish to translate. The bot passes the phrase on to the Web service, along with the language preferences chosen by the user. The bot then sends the translation provided by the Web service back to the user.

We have added functionality to this bot so that the user can change their language preferences at any time. The bot will remember the user’s current preferences for the duration of their chat session. The user’s preferences could of course be stored persistently, but we have left this as an exercise for the reader.

The Translator Bot is implemented by the TranslatorBot class, which extends the SametimeBot abstract class. As the majority of the code is very similar to that in the previous section, we only detail the changes here.

SametimeBot

To allow for the user’s language preferences to be changed during their chat session, we add a Hashtable object to the SametimeBot class:

```
protected Hashtable imState = new Hashtable();
```

We change the doTextReceived() method to include a reference to the Hashtable as one of its parameters. This gives the TranslatorBot subclass access to the Hashtable and its contents:

```
abstract protected String doTextReceived(ImEvent event, String strReceivedText,
Hashtable imState);
```

We also modify the `imClosed()` method to remove the user's `Im` object from the `Hashtable`, if it still contains it (see Example 3-16). It is at this point that you could implement logic to store the user's preferences persistently, if required.

Example 3-16 Modified `imClosed()` method

```
public void imClosed(ImEvent event)
{
    Im im = event.getIm();
    Im currentIm = null;
    for (int i = 0; i < m_imOpened.size(); i++)
    {
        currentIm = (Im) m_imOpened.elementAt(i);
        if (currentIm.getPartner().getId().equals(im.getPartner().getId()))
        {
            // If the Hashtable still contains the Im object
            if (imState.contains(im))
            {
                // Remove it
                imState.remove(im);
            }
            m_imOpened.removeElement(im);
            im.close(0);
            im.removeImListener(this);
            break;
        }
    }
}
```

TranslatorBot

The `TranslatorBot` class extends the `SametimeBot` class and overrides the `doTextReceived()` method. This method is called when the user sends a message to the bot, and is indicative of one of four possible events:

1. The user is sending the details of the language they are using.
2. The user is sending the details of the language in which they wish to receive the translation.
3. The user is sending a word or phrase to be translated.
4. The user is requesting to change their language preferences.

Each one of these events corresponds to a particular 'state' that the user is in. This state is stored in the `Hashtable` created in the `SametimeBot` superclass. When the `doTextReceived()` method is called (Example 3-17 on page 76), the `TranslatorBot` retrieves the user's current state from the `Hashtable` and responds accordingly.

Example 3-17 TranslatorBot doTextReceived() method

```
protected String doTextReceived(ImEvent event, String receivedText, Hashtable
imState)
{
    Im im = event.getIm();
    String previousState = "";
    String returnText = "";

    previousState = (String)imState.get(im);

    if (receivedText.toUpperCase().compareTo("CHANGECHOICES") == 0)
    {
        // Remove Im object from Hashtable
        imState.remove(im);
        // Prompt user to enter new preferences
        returnText = "Please enter the 2 digit code for the language you are
using.";
    }
    else if (previousState == null)
    {
        // First submission from user - language they are using
        imState.put(im, "translate1" + receivedText);
        returnText = "Please enter the 2 digit code for the language you wish to
receive. If you wish to change your language choices at any time, type
CHANGECHOICES";
    }
    else if
(previousState.toLowerCase().substring(0,10).compareTo("translate1") == 0)
    {
        // Second submission - language user wishes to receive
        String userLanguage = previousState.toUpperCase().substring(10,12);
        imState.put(im, "translate2" + userLanguage +
receivedText.toUpperCase());
        returnText = "Please enter the phrase you wish translating";
    }
    else
    {
        // Parse out language preferences
        String userLanguageFrom = previousState.toUpperCase().substring(10,12);
        String userLanguageTo = previousState.toUpperCase().substring(12,14);
        // Call web service with phrase and preferences
        try
        {
            returnText = doTranslate(receivedText, userLanguageFrom,
userLanguageTo);
        }
        catch (Exception e) {}
    }
}
```

```
    return returnText;
}
```

If the user is requesting a translation, the `doTextReceived()` method calls the `doTranslate()` method, passing in the phrase to be translated and the user's language preferences. The `doTranslate()` method contains code to contact the BabelFish Web service and return the translated phrase.

Figure 3-13 shows the bot responding to the `imReceived()` event and sending back a greeting message.



Figure 3-13 Bot responds to `imReceived()` event with custom greeting

When the user has entered the codes for the language she is using and the language she wishes to receive the translation in, the bot begins translating her phrases, as shown in Figure 3-14 on page 78.



Figure 3-14 Bot translates phrases according to user's preferences

At any time, the user can type the keyword CHANGECHOICES to reset her preferences. Her `Im` object is removed from the `Hashtable`, resetting her session state back to the beginning. The bot then prompts her to specify her new language preferences, as shown in Figure 3-15 on page 79.

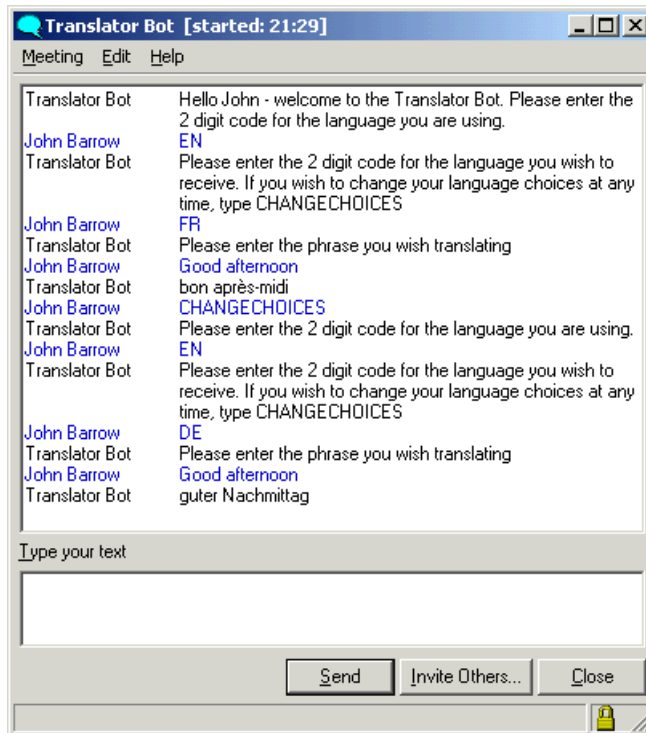


Figure 3-15 Bot responds with translation using new language preferences

The three bots described in this section are relatively simple, but show a general abstract class around which bots can be developed. In the next section, we discuss various enhancements to the ideas already outlined.

3.4 Enhancing the bot framework

The Echo Bot, FAQ Bot, and Translator Bot examples have shown you how to create bots using the Java Client Toolkit. One is a Java application, the other two Java servlets.

The FAQ Bot and Translator Bot show how you can create a framework bot class that implements all the functionality that every bot will need: the ability to log in to a Sametime server, to register for instant messages, and to respond to user interaction. You are then free to develop custom subclasses that implement the application-specific logic that you want your bot to exhibit.

The degree to which you enhance this basic framework is entirely up to you:

- ▶ You could use a database or XML file to hold the configuration settings for your bot, so there would be no need to hardcode server names, user names, or passwords.
- ▶ You could create a server application using the Community Server Toolkit to control your bots. It could be a Mux application that performed a light login for each of the bots under its control. It could read a configuration file containing information about the bots and then dynamically load the class files, acting as a virtual bot server. The server application could then be implemented as a Windows NT/2000 service using the JavaLauncher tool. For more details on developing server applications and using the JavaLauncher tool, consult *Working with the Sametime Community Server Toolkit*, SG24-6667.
- ▶ You could implement logic in your bot so that it would only become active at certain times, for example, a company news service bot could become active only when there was a new piece of company news; at other times, it could be logged out, away, or in a do-not-disturb status.
- ▶ An extension to this idea is to set an alert in your Connect client to alert you as soon as the bot became active, as shown in Figure 3-16.

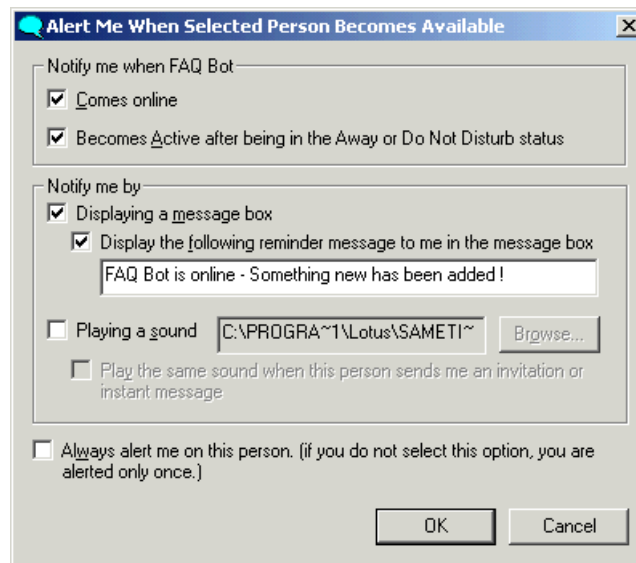


Figure 3-16 Alert triggered when FAQ Bot comes online or becomes active

- ▶ A bot has an awareness of the whole Sametime community and can tell if a specific user is online. A whole host of alert-type bots could be created to inform certain users of certain events, as long as they were online and available in Sametime.

- It is important to remember that bots are accessible to mobile users with cell phones or PDAs as well as to users connected to the network using the Connect client if the Sametime Everyplace® product is used. No extra programming effort is required.



Web services

This chapter provides an overview of Web services and their usage with Sametime. We discuss how, by building Sametime Web services, you can make online presence awareness and instant messaging a part of any Web services-enabled system.

We show how applications created with the Java Client Toolkit can be turned into Web services using the tools provided by WebSphere Studio Application Developer 5.0.

We also show how these Web services can then be deployed to an application server, such as WebSphere Application Server, and accessed by any Web services client.

4.1 Overview of Web services

As you may already be well aware, Web services have gained tremendous interest and coverage in the IT industry in recent years. There have been many books published on the topic, so we do not attempt to cover the complete spectrum of Web services in this chapter. We do, however, give a brief introduction, so you may gain an understanding as to why you may want to use Web services.

4.1.1 What is a Web service?

A widely accepted definition for a Web service is:

“A self-contained, self-describing, modular application that can be published, located and invoked across the Web.”

Web services enable the creation of applications that navigate, discover, and use other applications, in much the same way that people navigate, discover, and use Web sites and Web-based business applications.

With Web services, any business system can expose its underlying functionality in new ways. A Web services layer allows developers to easily and dynamically incorporate that system's functionality in any application, regardless of the language it is written in or the platform it runs on. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service. The Web services model shifts the focus to integrating applications, independent of platform, language, or data structure.

4.1.2 Web service fundamentals

Web services are built on the foundation of two technologies:

- ▶ XML: The universal language for data description. XML's separation of content and presentation allows data to be manipulated in a reliable, automated way. Instructions or content represented by XML can be understood by any application that supports XML.
- ▶ SOAP: The Simple Object Access Protocol uses XML messages to invoke remote methods. A Web service could interact with remote machines through HTTP's POST and GET methods, but SOAP is much more robust and flexible.

In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response.

On the server side, a Web server and, for Java-based Web services, a servlet engine is all that is required. On the client side, a programming language with XML and HTTP client support is all that is needed. It is possible to create a Web service from an existing application without writing a single line of code, as 4.4, “Creating the UserStatus Web service” on page 97 illustrates.

Web services are self-describing: neither the client nor the server needs to know anything besides the format and content of the request and response messages. How the actual service works at the application level is irrelevant to the client, and how the client generates the request and what it does with the response is irrelevant to the server.

You can generate an XML-based description of the Web service that explains how it can be accessed. The Web Services Description Language (WSDL) allows Web services-enabled systems to tell each other what capabilities they have, and how to programmatically interact with them. Once a Web service is described in WSDL, developers can create Web service clients to interact with it.

You can also publish the Web service to a directory of Web services so that it can be discovered by other Web service clients. Universal Description, Discovery, and Integration (UDDI) is an XML-based directory or registry of Web services. UDDI provides a place for you to list your Web service and to find out about others. The combination of WSDL and UDDI means Web service description and discovery can be automated, allowing clients to bind to dynamically-discovered services.

With SOAP, you can send a request to another application, for example, to return the status of an item from a shipping system. You get the results back in a set format that you can then process. To publish the shipping system as a Web service, you register the application with an application server that supports the SOAP protocol (which runs as an HTTP extension). You can generate a WSDL description of the application that explains how it can be accessed. You can then publish that description to a UDDI directory.

This section has been deliberately brief: we encourage you to gain a more in-depth understanding of Web services by consulting the Redbooks *WebSphere Version 5 Web Services Handbook*, SG24-6891 and *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292.

4.2 Sametime Web services

A Sametime Web service is simply a Web service layer that exposes the functionality of a Sametime application. Once you expose a Sametime application as a Web service, you give any client capable of submitting an HTTP

POST request the ability to interact with your Sametime application, regardless of the language or platform that the client uses.

For example, imagine a Sametime application written using the Java Client Toolkit. This could be exposed as a Web service and hosted on an application server, such as WebSphere Application Server or Jakarta Tomcat. A client written in Visual Basic or .NET could use this Web service to give itself online awareness and instant messaging capabilities.

In the next section, we show a Sametime application built with the Java Client Toolkit that returns the online status of a user in the Sametime community. We then turn that application into a Web service using the tools that are included with WebSphere Studio Application Developer. Finally, we show how you can deploy the Web service to WebSphere Application Server so that it may be accessed by Web service clients of any description. We show how you can access the Web service from a sample JSP page created by WebSphere Studio Application Developer.

4.3 Building the UserStatus application

In this section, we build a Java application using the Client Toolkit that, for a given Sametime user, returns a numerical value indicating their current status, that is, Active, Away, Offline, and so on.

The UserStatus application performs the following tasks as part of its execution:

1. Logs in to the Sametime server.
2. Resolves the user name, passed as a parameter to the application, to a Sametime user STUser object.
3. Gets and returns the current status of that STUser object in the Sametime community.
4. Logs out of the server.

4.3.1 The UserStatus application

The class UserStatus starts by importing a number of Sametime-specific packages, as shown in Example 4-1.

Example 4-1 Package imports for UserStatus class

```
import com.lotus.sametime.awareness.*;
import com.lotus.sametime.community.*;
import com.lotus.sametime.core.comparch.*;
import com.lotus.sametime.core.constants.*;
```



```
import com.lotus.sametime.core.types.*;
import com.lotus.sametime.lookup.*;
```

The class implements three Sametime listener interfaces: LoginListener, ResolveListener, and StatusListener.

The LoginListener interface allows the application to receive LoginEvent events, of which there are two: loggedIn() and loggedOut(). The application is notified by the Community Service when the login process has finished by calling one of these two methods.

The ResolveListener interface allows the application to receive ResolveEvent events that are fired by the Lookup Service in response to requests to resolve user names as Sametime objects. The ResolveListener interface implements three methods: resolved(), resolveConflict(), and resolveFailed(). If the user name is successfully resolved with only one match in the directory, the resolved() method is called and we can get a handle on the Sametime STUser object for that user.

The StatusListener interface implements two methods: groupCleared() and userStatusChanged(), and allows the application to receive StatusEvent events about the STUser object resolved by the Lookup Service, from which its current status can be derived.

The UserStatus class defines a number of variables for the Sametime objects it uses, the name of the user to search for and their status, and some booleans to handle event timing (see Example 4-2 for more details).

Example 4-2 Interfaces implemented by UserStatus class and variable declarations

```
public class UserStatus implements LoginListener, ResolveListener,
StatusListener
{
    private STSession m_session;
    private CommunityService m_communityService;
    private LookupService m_lookupService;
    private AwarenessService m_awarenessService;
    private Resolver m_resolver;
    private WatchList m_watchList;
    private STUser m_requestedUser;

    private String m_strUserName;
    private short m_shUserStatus;
    private boolean m_bSetup;
    private boolean m_bResolved;
    private boolean m_bFinished;
```

The application contains a `main()` method that takes one parameter: a `String` representing the user whose status we are interested in. The `main()` method instantiates a new `UserStatus` object and calls its `getUserStatus()` method (Example 4-3), passing in the `String` parameter. The `getUserStatus()` method returns a Java short object representing the user's status:

- ▶ 32: User is online and active.
- ▶ 96: User is online but away.
- ▶ 128: User is online but in 'Do not disturb' status.
- ▶ 0: User is offline.

Note: You can find all of the official Sametime status in the Javadoc reference in the `STUserStatus` class description.

Example 4-3 UserStatus main() method

```
public static void main(String[] args)
{
    if(args.length != 1)
    {
        System.out.println("Usage: UserStatus userToFind");
        System.exit(0);
    }

    short shUserStatus = new UserStatus().getUserStatus(args[0]);
    System.out.println("Status for user " + args[0] + ": " + shUserStatus);
}
```

The `getUserStatus()` method (Example 4-4 on page 89) creates a new Sametime session object and loads the required components into it. It then starts the session.

Attention: For a more detailed discussion of this process, see 3.2.1, "Creating a Sametime session" on page 52.

The method gets references to both the `CommunityService` and `LookupService` components before attempting to log in to the Sametime server. It waits until it has successfully logged in to the server and set up the Sametime objects it needs or until it exceeds its maximum number of setup attempts. If the setup completes successfully, the `resolve()` method is called and finally the value of the `m_shUserStatus` variable is returned. This value gives the current status for the requested user.

Example 4-4 *UserStatus* *getUserStatus()* method

```
public short getUserStatus(String userToFind)
{
    m_strUserName = userToFind;
    m_bSetup = false;

    try
    {
        m_session = new STSession("UserStatus" + this);
    }
    catch (DuplicateObjectException doe)
    {
        doe.printStackTrace();
        return -1;
    }

    m_session.loadSemanticComponents();
    m_session.start();

    m_communityService = (CommunityService)
m_session.getCompApi(CommunityService.COMP_NAME);
    m_lookupService = (LookupService)
m_session.getCompApi(LookupService.COMP_NAME);
    login();

    int setupAttempts = 0;
    do
    {
        if(++setupAttempts > 5000)
        {
            System.err.println("Setup attempt timed out");
            m_bSetup = true;
        }

        try
        {
            Thread.sleep(5);
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
    while (m_bSetup == false);

    if(isLoggedIn() == false)
    {
```

```

        System.err.println("Error: cannot login to Sametime server");
        return -1;
    }

    resolve();

    return m_shUserStatus;
}

```

The login() method and its helper, the isLoggedIn() method (Example 4-5), attempt to log the application in to the Sametime server. The application waits until login is successful or the number of login attempts is exceeded. If the login attempt is unsuccessful, the Sametime session is stopped and unloaded. If the attempt is successful, the isLoggedIn() method is called.

Example 4-5 UserStatus login() and isLoggedIn() methods

```

private void login()
{
    m_communityService.addLoginListener(this);
    m_communityService.loginByPassword("<servername>", "<servicename>",
"<servicepassword>");

    int loginAttempts = 0;
    boolean loginExpired = false;
    do
    {
        if (++loginAttempts > 1000)
        {
            loginExpired = true;
        }

        try
        {
            Thread.sleep(5);
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
    while (isLoggedIn() == false && loginExpired == false);

    if(isLoggedIn() == false)
    {
        m_session.stop();
        m_session.unloadSession();
        m_bSetup = true;
    }
}

```

```

    }
}

private boolean isLoggedIn()
{
    if(m_communityService == null)
    {
        return false;
    }
    return m_communityService.isLoggedIn();
}

```

Attention: For a more detailed discussion of the login process, see 3.2.2, “Logging in to the community” on page 53.

The `isLoggedIn()` method (Example 4-6) is called upon successful login to the Sametime server. The application creates a `Resolver` object from the `LookupService` component. It uses the `Resolver` to resolve the `String` user name provided as a parameter to the application into a Sametime `STUser` object. The application also creates a `WatchList` object from the `AwarenessService` component. It adds itself as `StatusListener` to the `WatchList` object so it can receive events whenever the status of the watched user changes. The boolean `m_bSetup` is then changed so the `getUserStatus()` method can continue.

Example 4-6 UserStatus isLoggedIn() method

```

public void isLoggedIn(LoginEvent event)
{
    System.out.println("isLoggedIn()");
    m_resolver = m_lookupService.createResolver(false, false, true, false);
    m_awarenessService = (AwarenessService)
m_session.getCompApi(AwarenessService.COMP_NAME);
    m_watchList = m_awarenessService.createWatchList();
    m_watchList.addStatusListener(this);
    m_bSetup = true;
}

```

Once the `m_bSetup` flag is switched, the `resolve()` method (Example 4-7 on page 92) is called. The application adds itself as a `ResolveListener` to the `Resolver` object. It then calls the `resolve()` method of the `Resolver` object, passing it the user name of the user we are interested in. The application waits to be notified on the success of the resolve process. It does this by receiving one of the three events defined in the `ResolveListener` interface. Upon completion, the application removes itself as a `ResolveListener`. It then waits on confirmation that the requested user’s status has been obtained. Once it receives this confirmation, it removes itself as a `StatusListener` from, and then closes, the

WatchList object. Finally, it logs out of the Sametime server and stops and unloads the Sametime session.

Example 4-7 UserStatus resolve() method

```
private void resolve()
{
    m_bResolved = false;
    m_resolver.addResolveListener(this);
    m_resolver.resolve(m_strUserName);

    int resolveAttempts = 0;
    do
    {
        if(++resolveAttempts > 1000)
        {
            m_bResolved = true;
        }

        try
        {
            Thread.sleep(5);
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
    while (m_bResolved == false);

    m_resolver.removeResolveListener(this);

    m_bFinished = false;
    int finishAttempts = 0;

    do
    {
        if(++finishAttempts > 1000)
        {
            m_bFinished = true;
        }

        try
        {
            Thread.sleep(5);
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
```

```

    }
}
while (m_bFinished == false);

m_watchList.removeStatusListener(this);
m_watchList.close();

m_communityService.logout();
boolean logoutExpired = false;
int logoutAttempts = 0;

do
{
    if(++logoutAttempts > 1000)
    {
        logoutExpired = true;
    }

    try
    {
        Thread.sleep(5);
    }
    catch (InterruptedException ie)
    {
        ie.printStackTrace();
    }
}
while (isLoggedIn() == true && logoutExpired == false);

m_session.stop();
m_session.unloadSession();
}

```

The ResolveListener interface defines three methods: the UserStatus application is only interested in the resolved() method (Example 4-8). In this method, the String user name has been resolved into an STUser object. The STUser object is the primary object used by Sametime to represent a valid, logged in user. The WatchList object uses the STUser object to monitor the object's status. The application adds the STUser object to the WatchList object and sets the m_bResolved variable so the resolve() method can continue.

Example 4-8 UserStatus resolved() method

```

public void resolved(ResolveEvent event)
{
    System.out.println("resolved()");
    if(event.getName().compareTo(m_strUserName) == 0)
    {

```

```
        m_requestedUser = (STUser) event.getResolved();
        m_watchList.addItem(m_requestedUser);
        m_bResolved = true;
    }
}
```

The StatusListener interface defines two methods, only one of which the application is interested in: `userStatusChanged()` (Example 4-9). This method is called whenever the status of the watched user changes. This includes the first time the user is added to a WatchList object. We call the `getStatusType()` method of the STUserStatus object obtained from the StatusEvent object to obtain the user's current status. We then set the `m_bFinished` variable so the `resolve()` method can continue.

Example 4-9 UserStatus userStatusChanged() method

```
public void userStatusChanged(StatusEvent event)
{
    System.out.println("userStatusChanged()");
    STWatchedUser[] watchedUser = event.getWatchedUsers();
    m_shUserStatus = watchedUser[0].getStatus().getStatusType();
    m_bFinished = true;
}
```

4.3.2 Running the UserStatus application

Once we have compiled the UserStatus application, we can run it using WebSphere Studio Application Developer's built-in test environment.

With the Java perspective open and the UserStatus.java file highlighted, click **Run** → **Run**, as in Figure 4-1 on page 95.

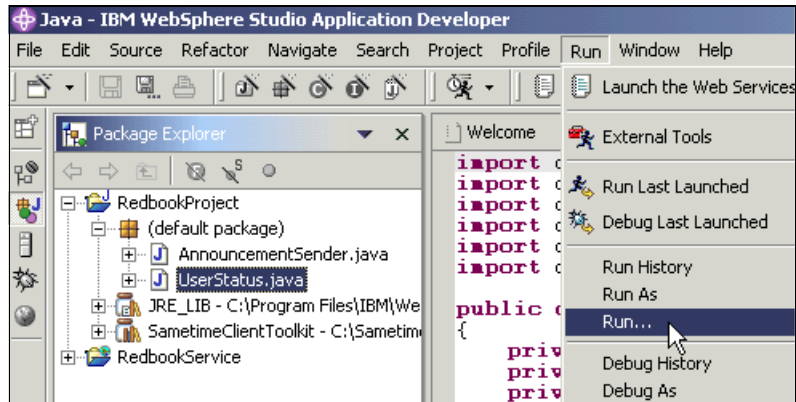


Figure 4-1 Running UserStatus application in WebSphere Studio Application Developer

In the dialog window that appears (Figure 4-2 on page 96), select **Java Application** in the left-hand window and click the **New** button. Click the **Arguments** tab, and enter a valid Sametime user's name in the Program arguments field. Then click **Run**.

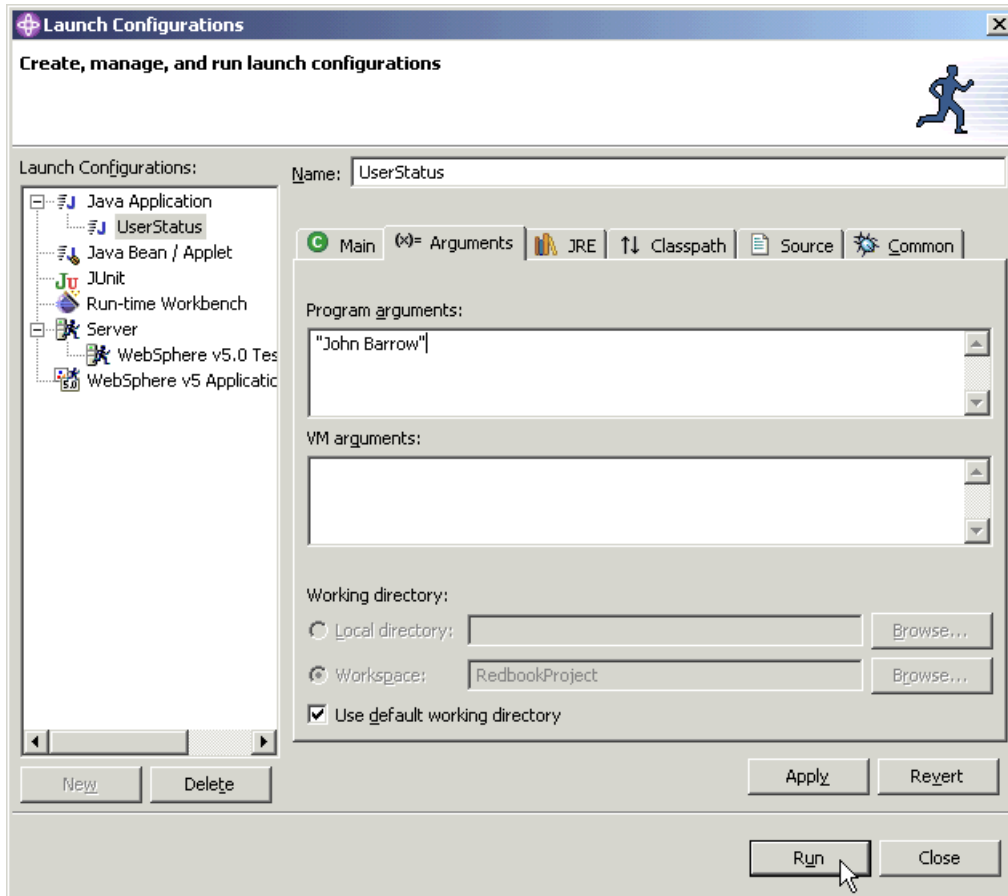


Figure 4-2 Launch Configurations dialog

The application will then run. If the user is currently active in Sametime, Figure 4-3 shows the console output in WebSphere Studio Application Developer.

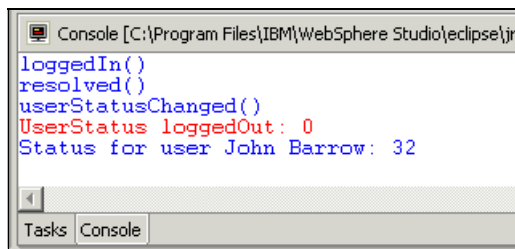


Figure 4-3 Console output for an active Sametime user

If the user is online, but in an 'away' status, you will see the output shown in Figure 4-4.

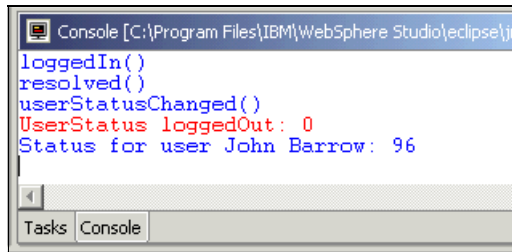


Figure 4-4 Console output for a Sametime user that is away

Now that we have built, compiled, and run our UserStatus application, the next step is to turn the application into a Web service.

4.4 Creating the UserStatus Web service

Using tools that come as part of WebSphere Studio Application Developer, it is possible to expose an existing Sametime application as a Web service without writing a single line of code. Using WebSphere Studio Application Developer's Web services wizards, we will expose a Web services layer on top of the UserStatus application.

The first step in creating an UserStatus Web service is to create a Web project to contain it:

1. Select **File -> New -> Project**.
2. Select **Web** in the left list box and then **Web Project** in the right list box. Click **Next**.
3. Give the project a name (we called ours RedbookService). Then click **Next** (Figure 4-5 on page 98).

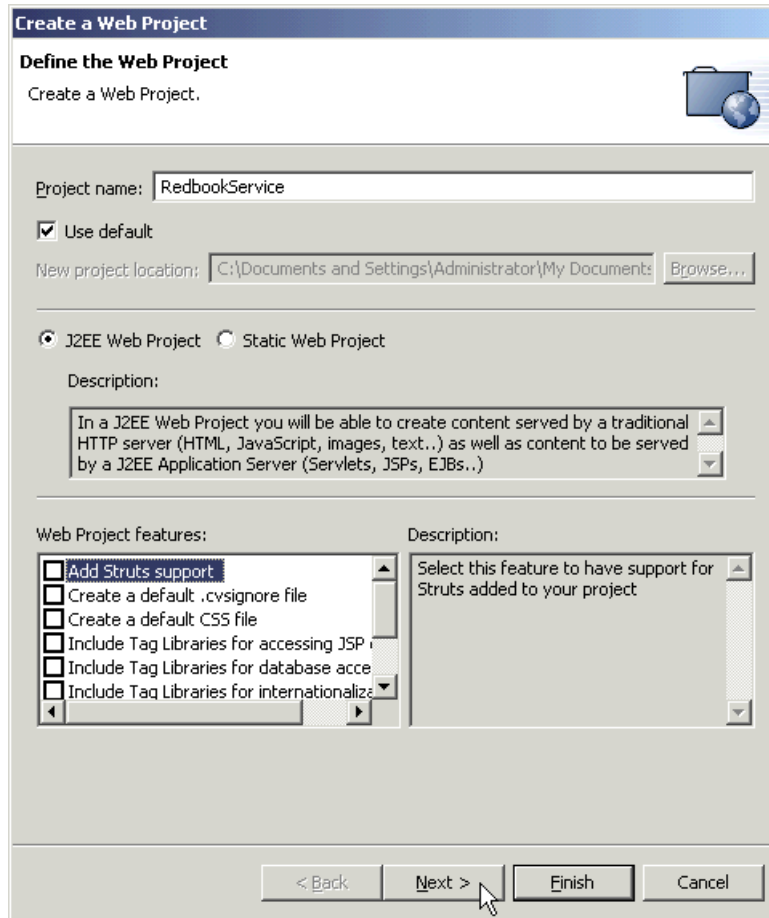


Figure 4-5 Defining the Web project

4. Create a new Enterprise Application project and give it a name (we called ours RedbookServiceEAR). An Enterprise Application project contains resources that are required to deploy an enterprise (J2EE) application. Associate it with this project by making sure the Context root points at the Web project. The context root is the Web application root, the top-level directory of the application when it is deployed to WebSphere Application Server. Then click **Finish** (Figure 4-6 on page 99).

Create a Web Project

J2EE Settings Page
Set the Enterprise Application project settings, context root, and J2EE level.

Enterprise application project: ☒ New ☐ Existing

New project name: RedbookServiceEAR

☒ Use default

New project location: C:\Documents and Settings\Administrator\My Documents

Context root: RedbookService

J2EE Level: 1.3

Description:
J2EE Level 1.3 includes a Servlet Specification level of 2.3 and a JSP Specification level of 1.2. Applications developed for this J2EE level typically target a WAS version 5.0 server.

< Back Next > **Finish** Cancel

Figure 4-6 Defining the EAR project

- Now we must add the Java Client Toolkit to our new project's classpath. Highlight the Web project, right-click, and then select **Properties** (Figure 4-7 on page 100).

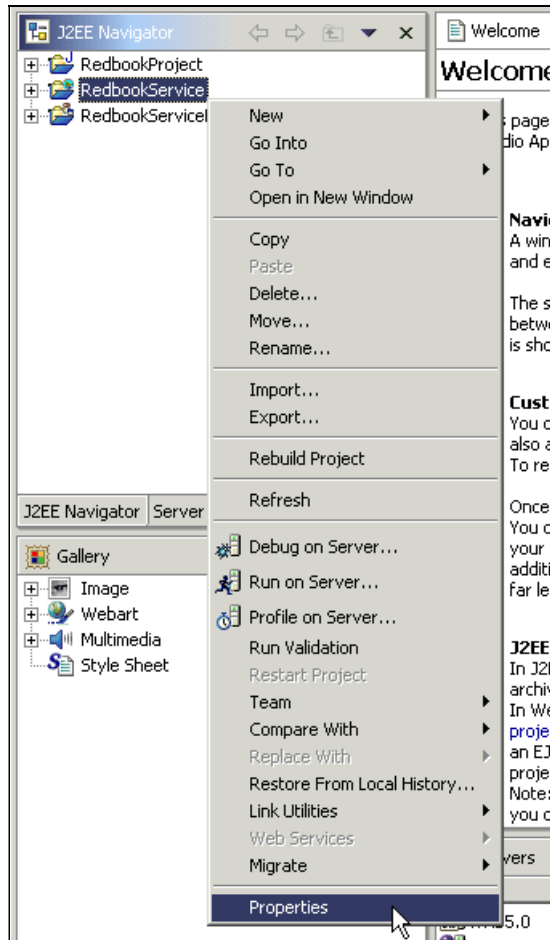


Figure 4-7 Selecting the project properties

6. Highlight the **Java Build Path** item in the left hand list. Select the **Libraries** tab and click on the **Add Variable** button (Figure 4-8 on page 101).

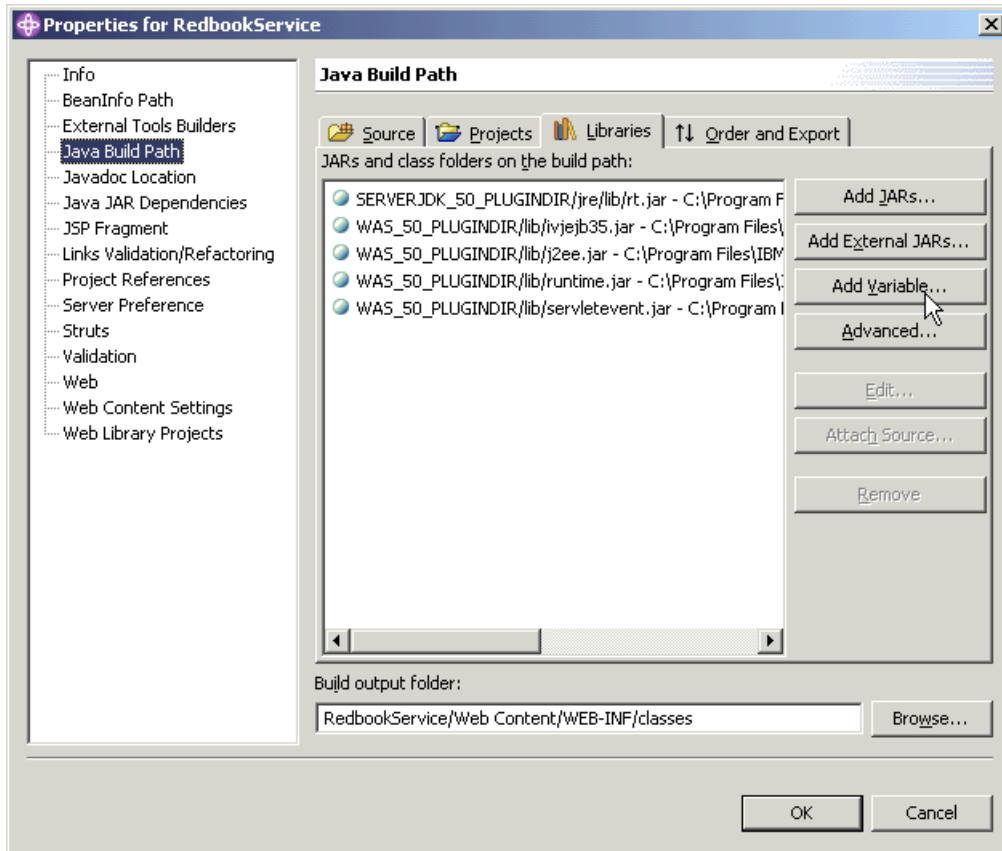


Figure 4-8 Java Build Path: Libraries tab

7. Select the variable for the Sametime Java Client Toolkit binaries. If you have not set up a variable for the binaries, see 2.3.1, “IBM WebSphere Studio Application Developer 5.0” on page 26. Click **OK** to continue (Figure 4-9 on page 102).

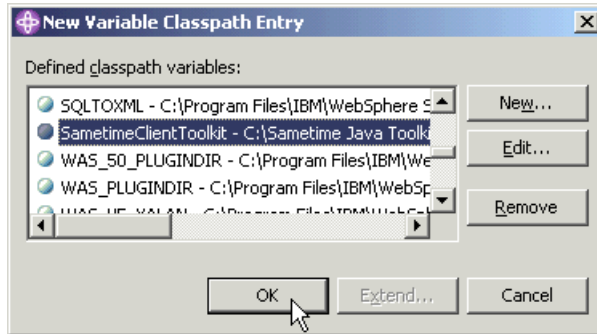


Figure 4-9 Adding the Java Client Toolkit variable

8. You will be returned to the Web perspective. Ensure that the Client Toolkit jar file has been included in the Libraries folder of your Web project (Figure 4-10).

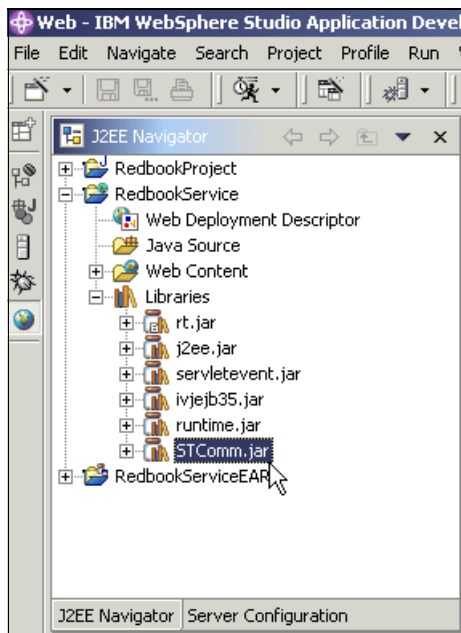


Figure 4-10 Client Toolkit jar added to project classpath

9. As we will be deploying our Web application to the WebSphere Application Server 5.0 test environment after we have created our Web service, we also need to add the Client Toolkit jar file to the Web Content\WEB-INF\lib folder as well. Highlight the lib folder, right-click, and select **Import**. Choose **File**

system and click **Next**, then browse to the location of the jar file and click **Finish**.

10. The next step is to import the UserStatus application we created in the last section. Highlight the Java Source folder, right-click, and select **Import** (Figure 4-11).

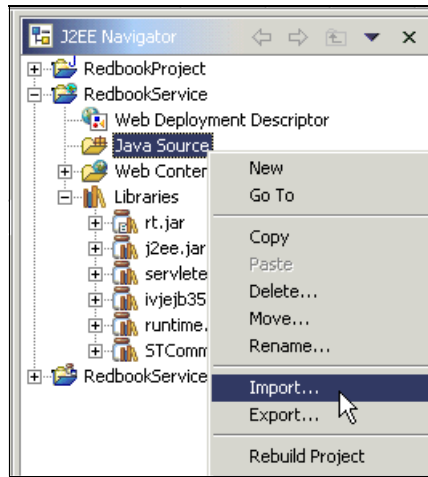


Figure 4-11 Import UserStatus application into RedbookService project

11. Select the file system in the import source list and click **Next**.
12. Navigate to the location of the UserStatus.java file created in the previous section, then click **Finish**. Verify that the file now appears under the Java Source folder (Figure 4-12 on page 104).

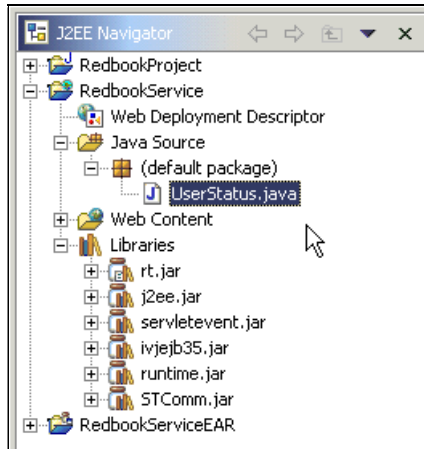


Figure 4-12 *UserStatus.java* file imported into *RedbookService* project

Now that we have the *UserStatus* application as part of the Web project, we can invoke WebSphere Studio Application Developer's Web services wizards to help us create our *UserStatus* Web service.

1. With the *UserStatus.java* file highlighted, right-click and select **New -> Other**.
2. Select **Web Services** in the left-hand list and **Web Service** in the right-hand list, then click **Next**.
3. Select **Java bean Web Service** as the Web service type, then click **Next**.
4. Leave the default settings and make sure the correct Web project is selected. Click **Next** (Figure 4-13 on page 105).

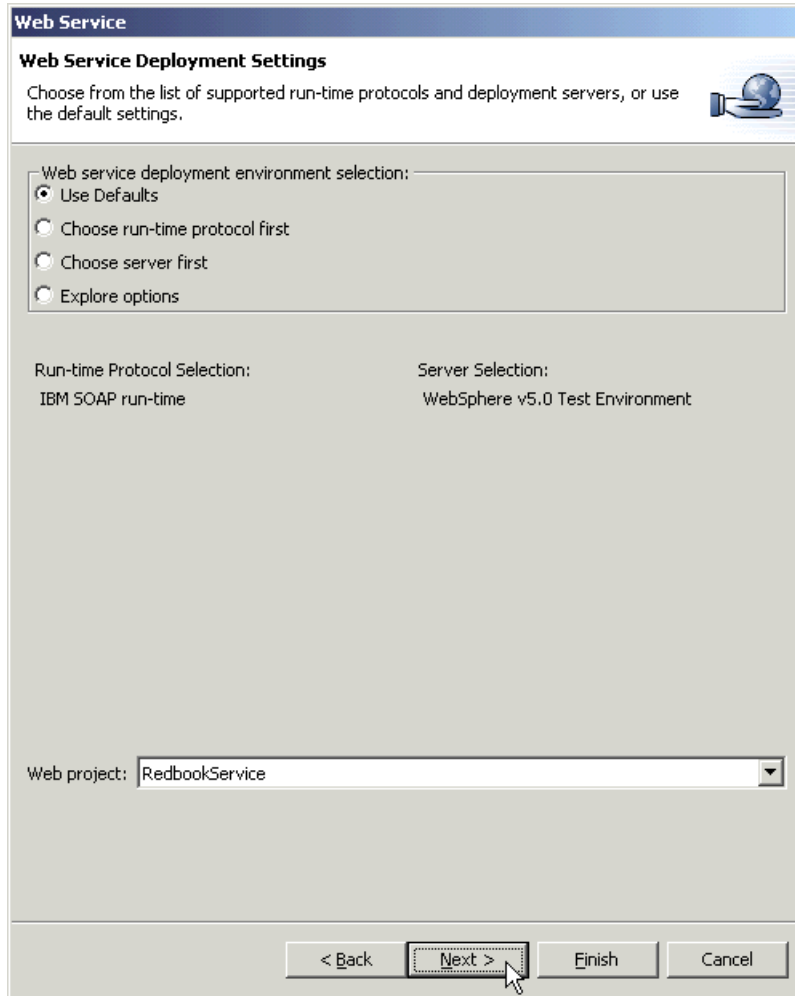


Figure 4-13 Web Service Deployment Settings dialog

5. Select UserStatus as the Java Bean, and click **Next**.
6. Change the Web service URI to something more easily remembered, like urn:UserStatus. A URI is a name that uniquely identifies a Web service to a client. Leave all the other settings as they are and click **Next** (Figure 4-14 on page 106).



The image shows a dialog box titled "Web Service". Inside, the section "Web Service Java Bean Identity" has the instruction "Configure the Java bean as a Web service." Below this, there are two input fields: "Web service URI:" with the value "urn:UserStatus" and "Scope:" with the value "Application". At the bottom, there are two unchecked checkboxes: "Use static methods" and "Use secure SOAP (WebSphere only)".

Figure 4-14 Web Service Java Bean Identity dialog

7. Figure 4-15 on page 107 shows a summary of the methods in our JavaBean. Deselect all the methods apart from `getUserStatus` (this is the only method we wish to expose to the Web service interface). Notice that it takes a `String` as a parameter and returns a `short`. Leave the other settings as they are; they allow us to select the encoding for our data. These are the rules for serializing data over the SOAP protocol. They tell the SOAP run-time environment how to translate from data structures constructed in Java into SOAP XML and vice versa. SOAP encoding is based on a simple type system, and literal XML encoding is based on an XML schema instance. We will use the default SOAP encoding. Click **Next**.

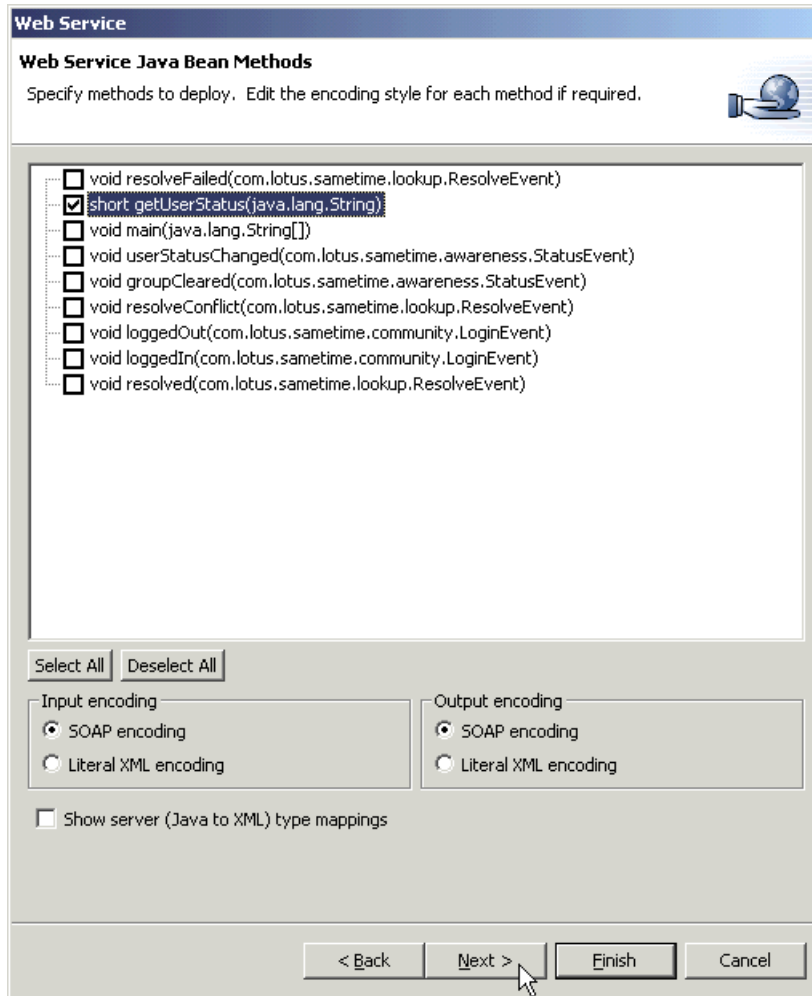


Figure 4-15 Web Service Java Bean Methods dialog

8. Select **Generate proxy** and **soap binding “UserStatusBinding”** of port **“UserStatusPort”**. This will generate a Web service proxy class for us. A proxy class makes it very easy for us to invoke a Web service. We simply instantiate the proxy class and invoke the desired method on it. The proxy class encapsulates the SOAP client programming APIs and the details of writing, receiving, and parsing SOAP XML documents. To keep all of our code in the one project, change the Project to be **RedbookService**. Click **Next** (Figure 4-16 on page 108).

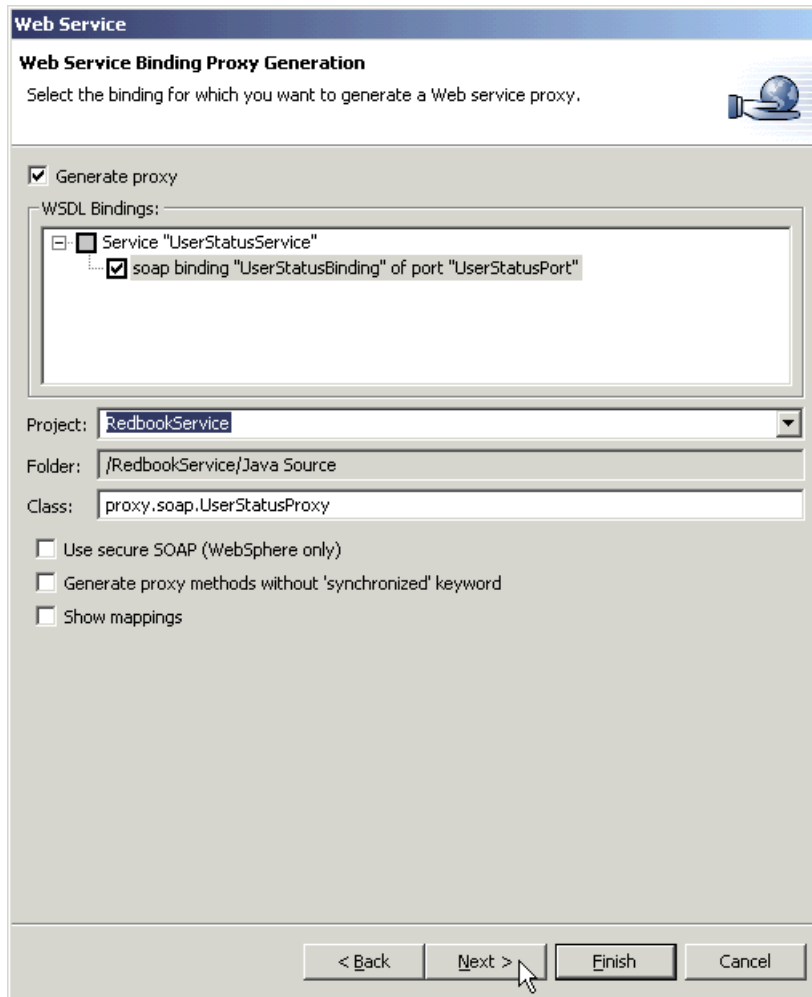


Figure 4-16 Web Service Binding Proxy Generation dialog

9. Select the **Test the generated proxy** check box and leave the Test facility as **Web service sample JSPs**. This tells WebSphere Studio Application Developer to generate a sample Web application consisting of some JSPs to test our Web service. These JSPs enable us to invoke the methods of the Web service and to view the results. The generated JSPs contain code to instantiate the Web service proxy class created in the previous step and to invoke the desired methods on it. Leave the other settings as they are and click **Finish** (Figure 4-17 on page 109).

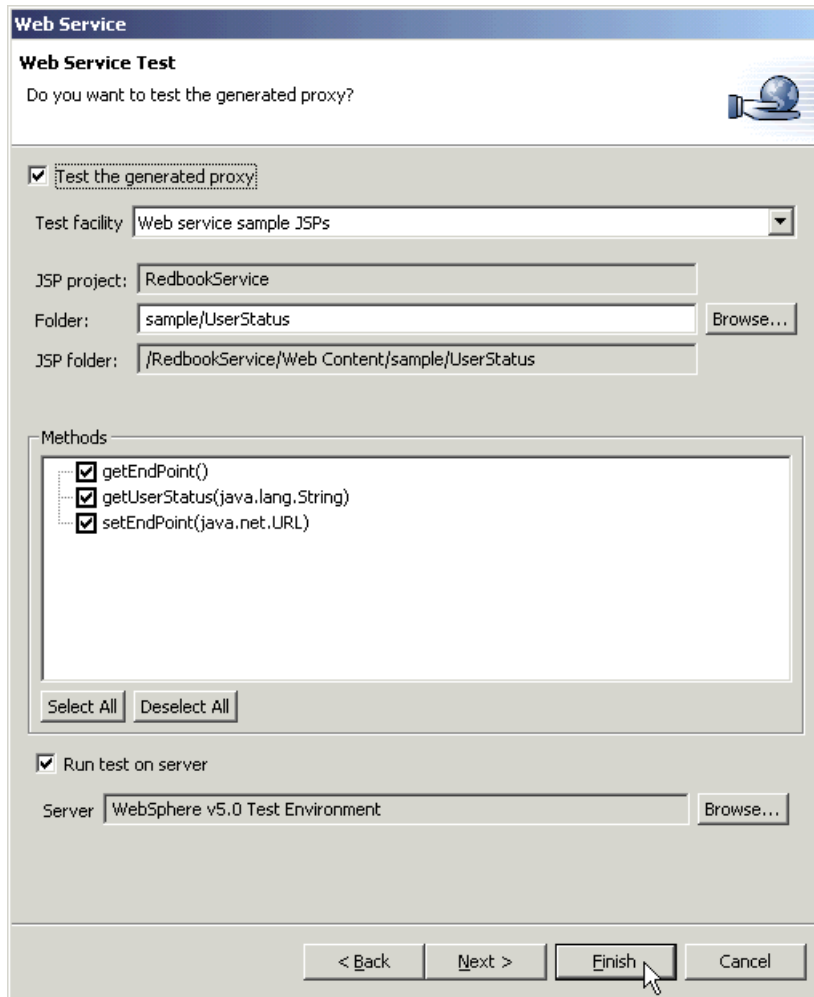


Figure 4-17 Web Service Test dialog

WebSphere Studio Application Developer will now deploy our Web service to its built-in WebSphere Application Server 5.0 test environment, and launch the TestClient.jsp file in its Web browser (see Figure 4-18 on page 111). Because we selected the option to test our Web service proxy, WebSphere Studio Application Developer created the four JSP files listed in Table 4-1 on page 110 in the Web Content\sample\User Status folder of the RedbookService project.

Table 4-1 Test JSP files created by WSAD's Web services wizards

File name	Description
TestClient.jsp	This is the frameset for the sample client and should be the URL you use to launch the Web service test page.
Method.jsp	This page provides a list of the available Web service methods; in our example, this includes getUserStatus(). When a link is selected, Input.jsp is invoked.
Input.jsp	This provides a form for the input parameters for each method defined in Method.jsp. In our example, this is a text field for the user name we are interested in. On submitting the form, Result.jsp is invoked.
Result.jsp	Contains an instance of the client proxy, and invokes it using the input parameters from the Input.jsp. This is where the user's current status is displayed.

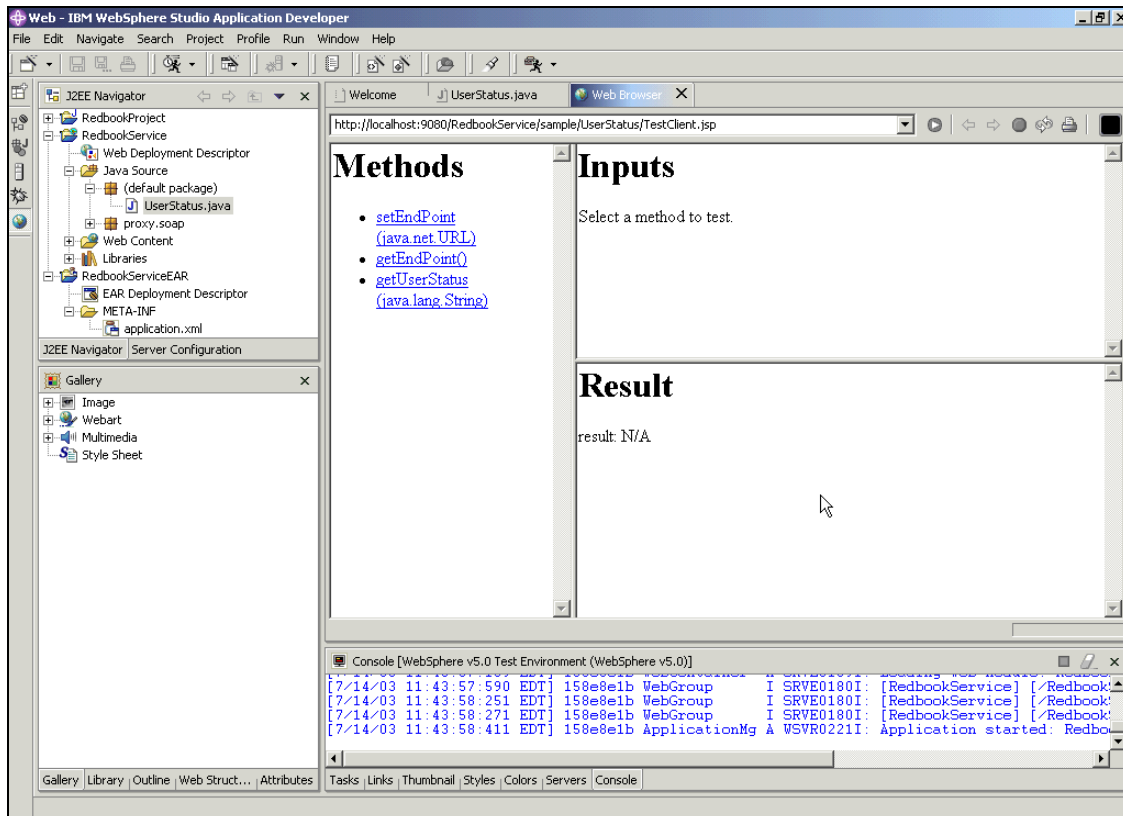


Figure 4-18 `TestClient.jsp` launched with WebSphere Studio Application Developer's internal Web browser

By clicking on the link for `getUserStatus()` in the `Method.jsp` frame, the `Input.jsp` file is called (Figure 4-19 on page 112).

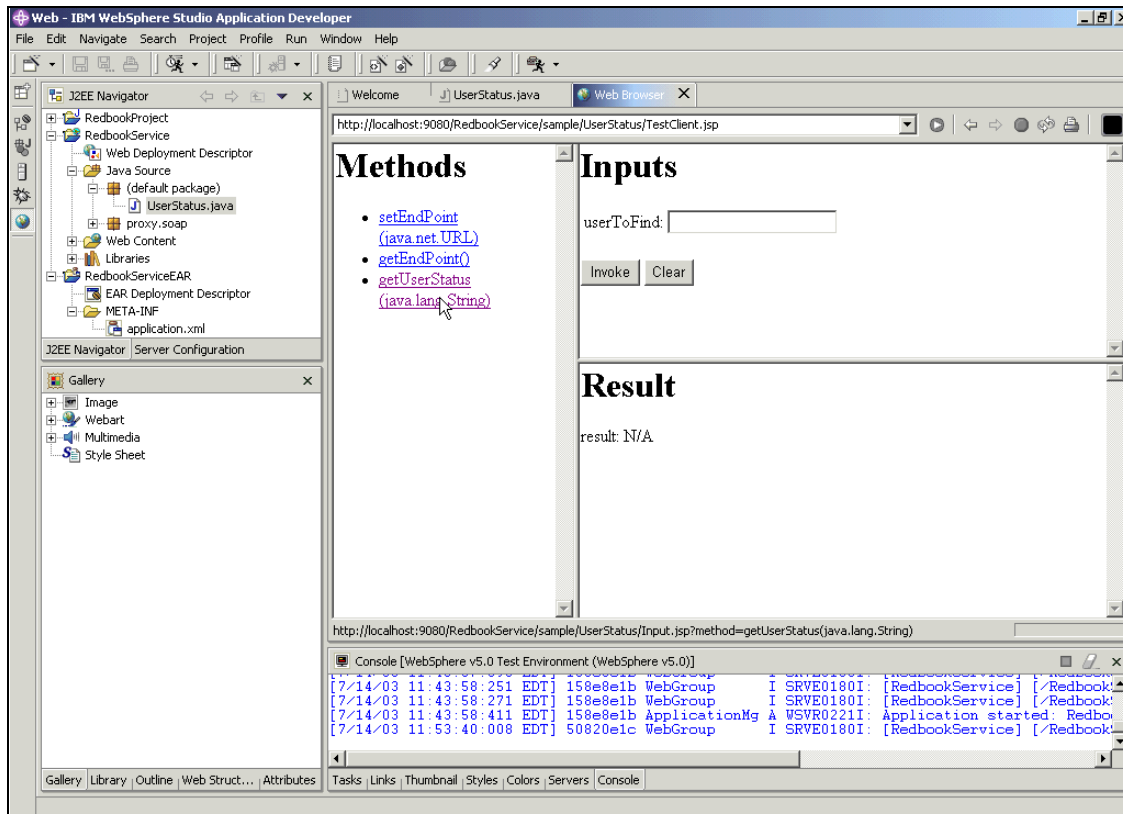


Figure 4-19 Clicking link for `getUserStatus()` loads `Input.jsp`

We can now enter a valid Sametime user's name in the text box and click on the **Invoke** button. If the user is online and active, `Result.jsp` shows the number 32 (Figure 4-20 on page 113).

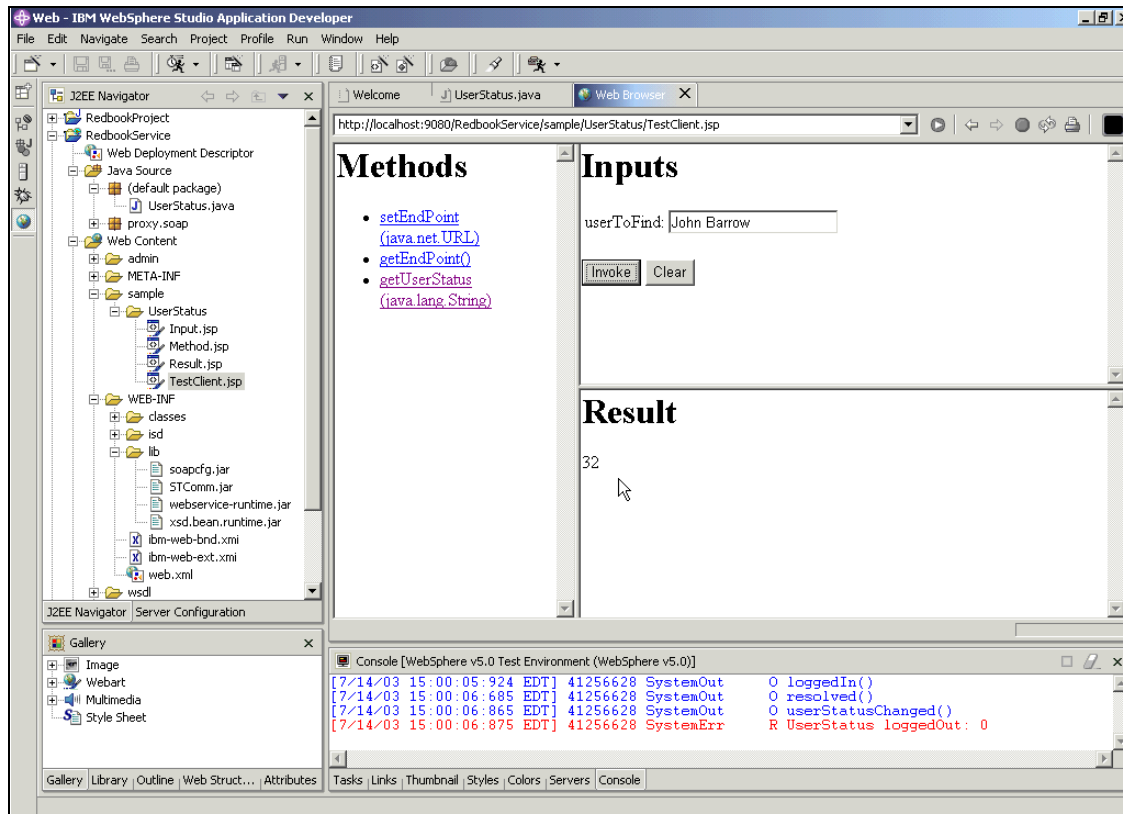


Figure 4-20 *Result.jsp* showing user is online and active

If the same user then changes his status to Away and we invoke the Web service again, *Result.jsp* changes to show the number 96 (Figure 4-21 on page 114).

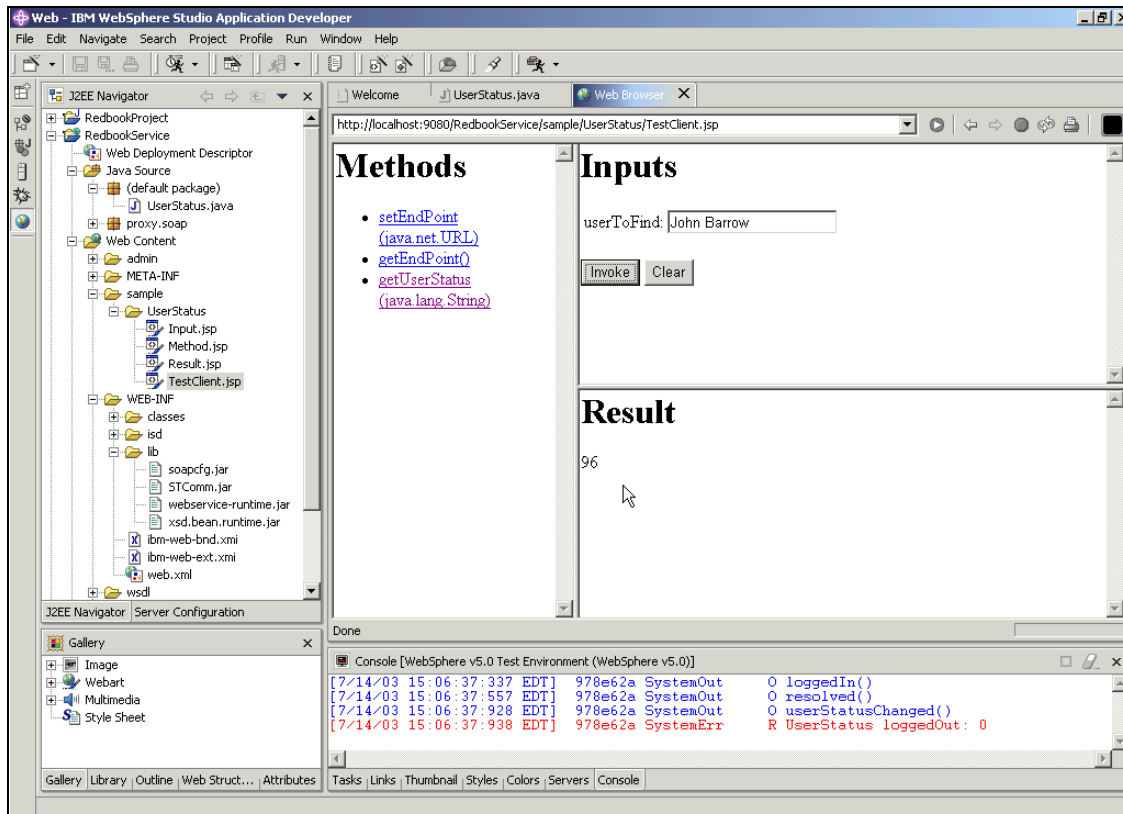


Figure 4-21 *Result.jsp* showing user is online but away

That concludes our look at creating a Sametime Web service. In this section, we have created a Web service interface to our UserStatus application, and we have demonstrated its use by calling it from a JSP page.

In the next section, we deploy the UserStatus Web service to WebSphere Application Server, so it can be accessed by any Web services client.

4.5 Deploying the UserStatus Web service

Once we have created our Sametime Web service, we can deploy it to an application server. For this example, we will deploy our UserStatus Web service to WebSphere Application Server 5.

Important: This section assumes you have a working version of WebSphere Application Server running. We do not cover how to install or configure WebSphere Application Server; if you need further guidance, please see the WebSphere Application Server InfoCenter, found at <http://www-3.ibm.com/software/webservers/appserv/infocenter.html>.

We deploy the UserStatus Web service by exporting the RedbookServiceEAR Enterprise Application project we created in the previous section. Before we can export it, however, we need to change a few WebSphere Studio Application Developer-specific entries.

1. The Java proxy class generated by WebSphere Studio Application Developer (Java Source\proxy.soap.UserStatusProxy.java) sets the URL for the Web service to:

```
http://localhost:9080/RedbookService/servlet/rpcrouter
```

If you deploy this Web service to another server, you need to replace localhost with the name of your server. In our case, we changed the following line in UserStatusProxy.java:

```
private String stringURL =  
"http://localhost:9080/RedbookService/servlet/rpcrouter";
```

to

```
private String stringURL =  
"http://t20-issluser:9080/RedbookService/servlet/rpcrouter";
```

The transport port for the application on WebSphere Application Server is 9080. You will need to check with your WebSphere Application Server administrator for the exact port number in your environment. Make sure you save your changes when you are done.

Figure 4-22 on page 116 gives an overview of these changes.

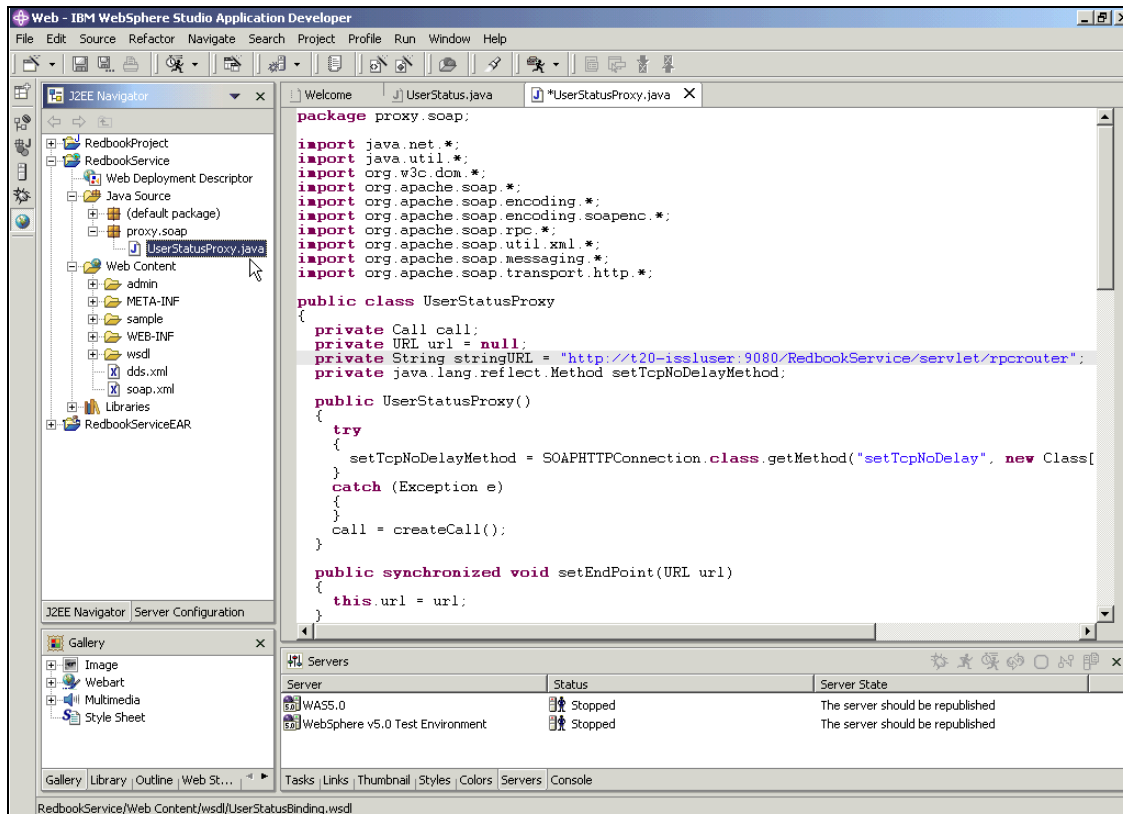


Figure 4-22 Change the default `UserStatusProxy.java` file created by WebSphere Studio Application Developer

2. We also need to change one of the Web Services Description Language (WSDL) files created by WebSphere Studio Application Developer. WSDL is a standard for describing networked, XML-based services. Its an XML text document that describes where a Web service is deployed and what operations it provides. Edit the file `Web Content\wsdl\UserStatusService.wsdl` and replace the one occurrence of `localhost` with your server name. Save your changes (see Figure 4-23 on page 117).

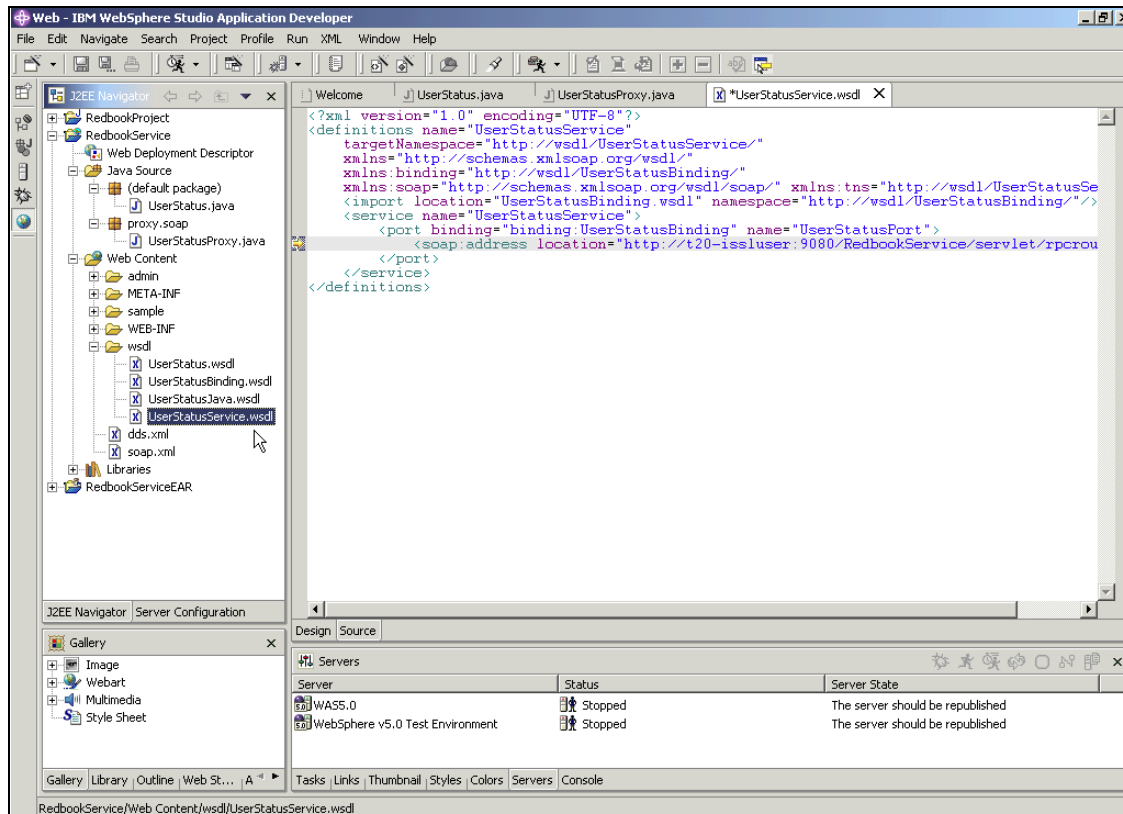


Figure 4-23 Replace the localhost value with your server name

We can now export our Enterprise Application project, ready for deployment to our WebSphere server:

1. Select **RedbookServiceEAR**, right-click, and select **Export**. Select **EAR file** from the list and click **Next**.
2. For **Where do you want to export resources to**, enter `C:\temp\RedbookServiceEAR.ear` or any other directory. Select **Finish**.

We are now ready to install our Enterprise Application. Launch the Administrative Console of WebSphere Application Server:

1. Expand the **Applications** item in the left-hand menu, and click the **Install New Application** link. Select **Local path** and browse to the location of the `RedbookServiceEAR.ear` file (`C:\temp\RedbookServiceEAR.ear` in our case) (Figure 4-24 on page 118).

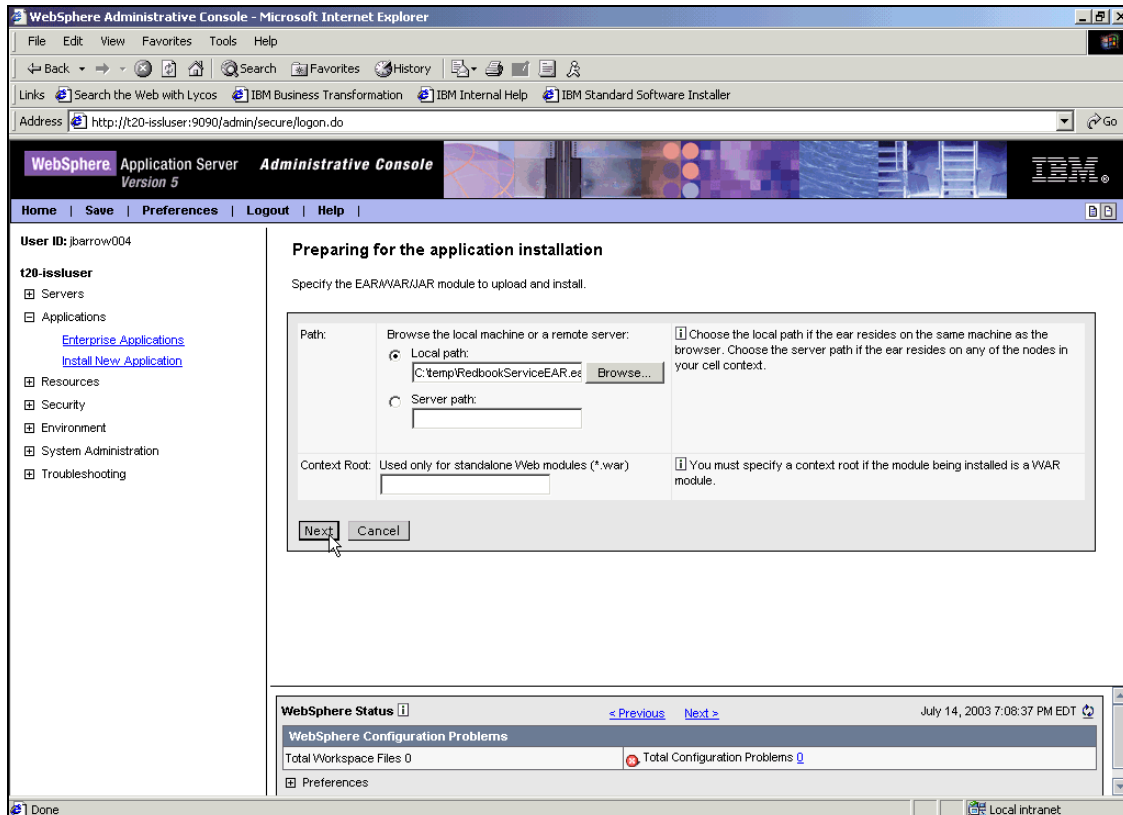


Figure 4-24 WebSphere Application Server Administrative Console: Install New Application

2. Accept the defaults on the next five screens. A confirmation screen (Figure 4-25 on page 119) then appears informing you that the Enterprise Application has been installed successfully. Click on the link to **Save to Master Configuration**.

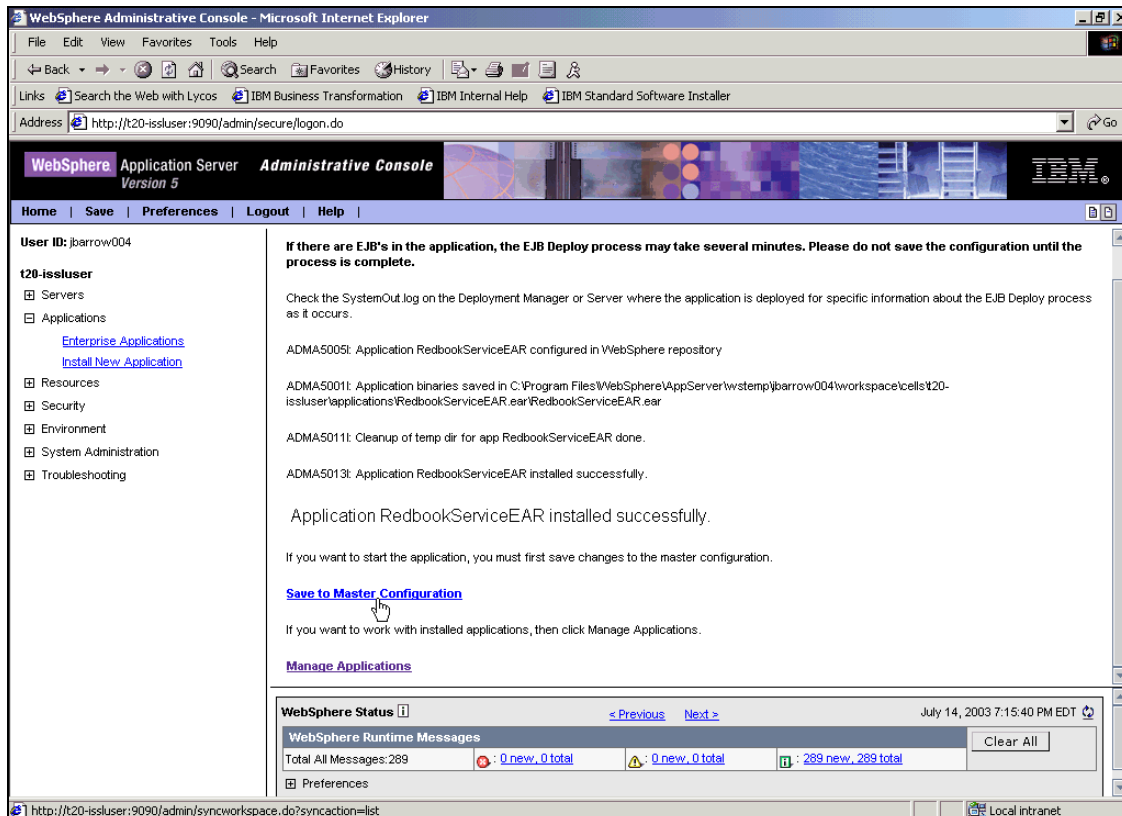


Figure 4-25 Confirmation screen following successful install of EAR

- On the **Save to Master Configuration** page, click the **Save** button.
- Click on **Enterprise Applications** in the left-hand menu.
RedbookServiceEAR now appears in the list, with its status marked as Stopped (Figure 4-26 on page 120).

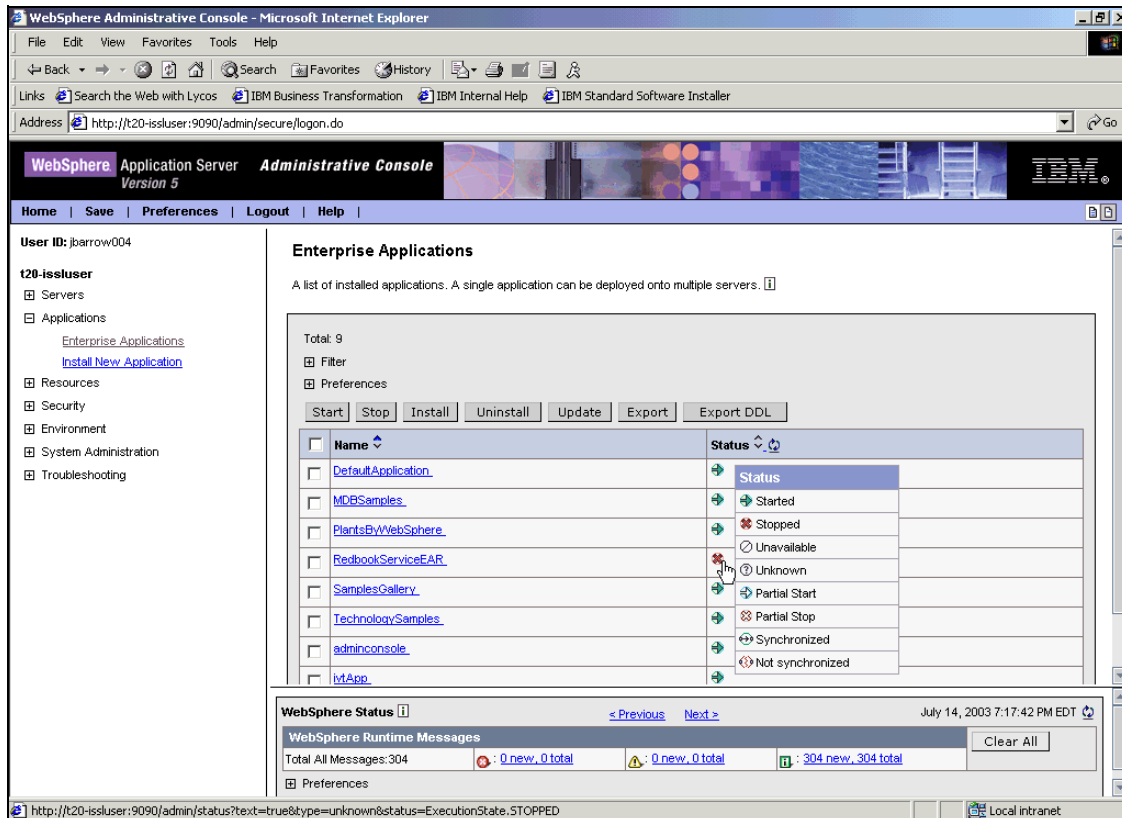


Figure 4-26 RedbookServiceEAR in list of installed Enterprise Applications

5. Start the RedbookServiceEAR Enterprise Application by selecting it and clicking the **Start** button (Figure 4-27 on page 121).

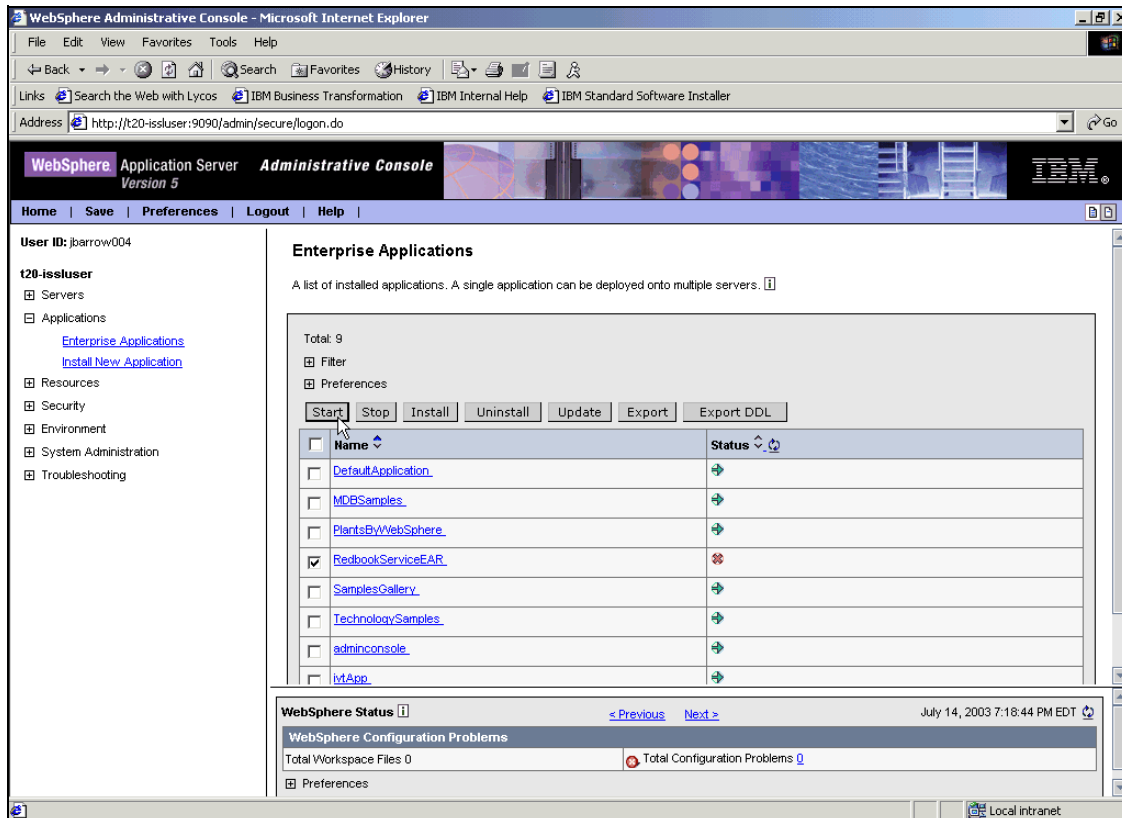


Figure 4-27 Start the RedBookServiceEAR Enterprise Application

6. A confirmation screen (Figure 4-28 on page 122) appears after the application has been started.

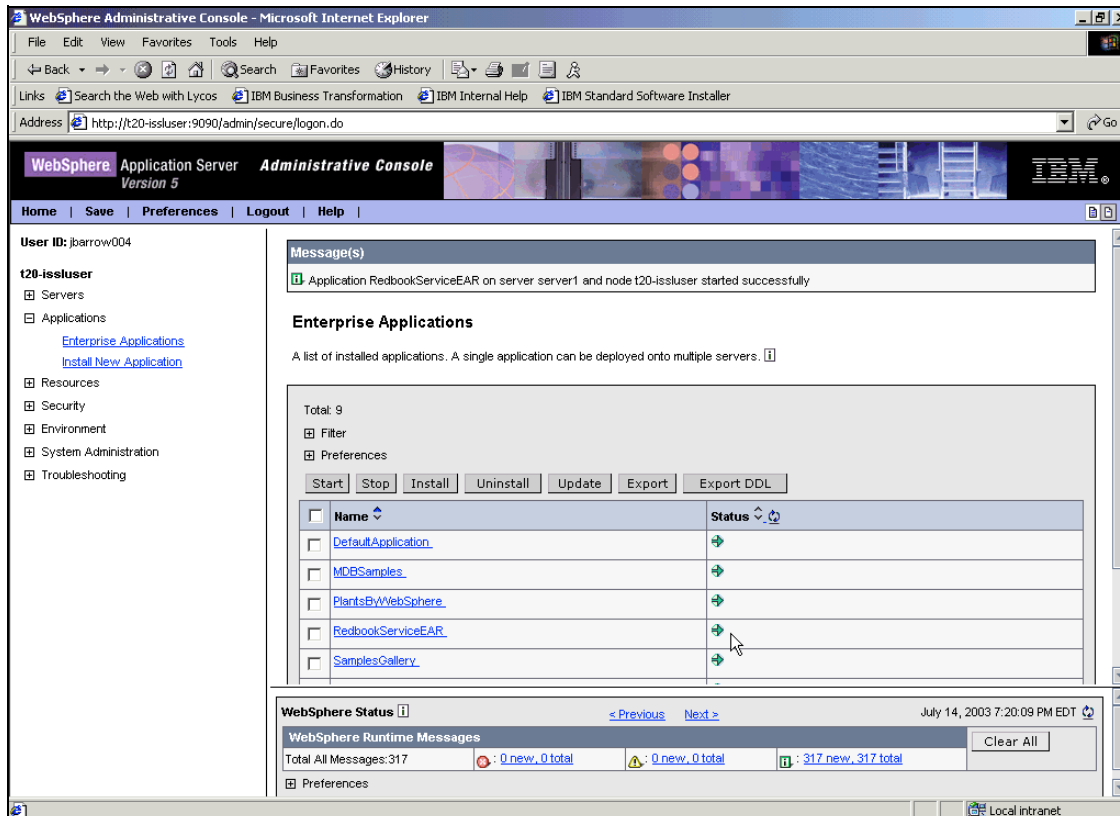


Figure 4-28 Confirmation screen following successful start of Enterprise Application

Now we have successfully deployed our Sametime Web service, we can go on and test it.

4.6 Testing the Sametime Web service

To test our Web service, we can call the sample JSP files that were included in our Enterprise Application deployment:

1. Open a Web browser and navigate to the URL of the Web service test client page. In our case, this is:

`http://t20-issluser/RedbookService/sample/UserStatus/TestClient.jsp`

You do not need to include the port number if WebSphere Application Server is on a different machine than the browser. You will see the TestClient.jsp frameset, with a link to the `getUserStatus()` method in the Methods frame (Figure 4-29 on page 123).

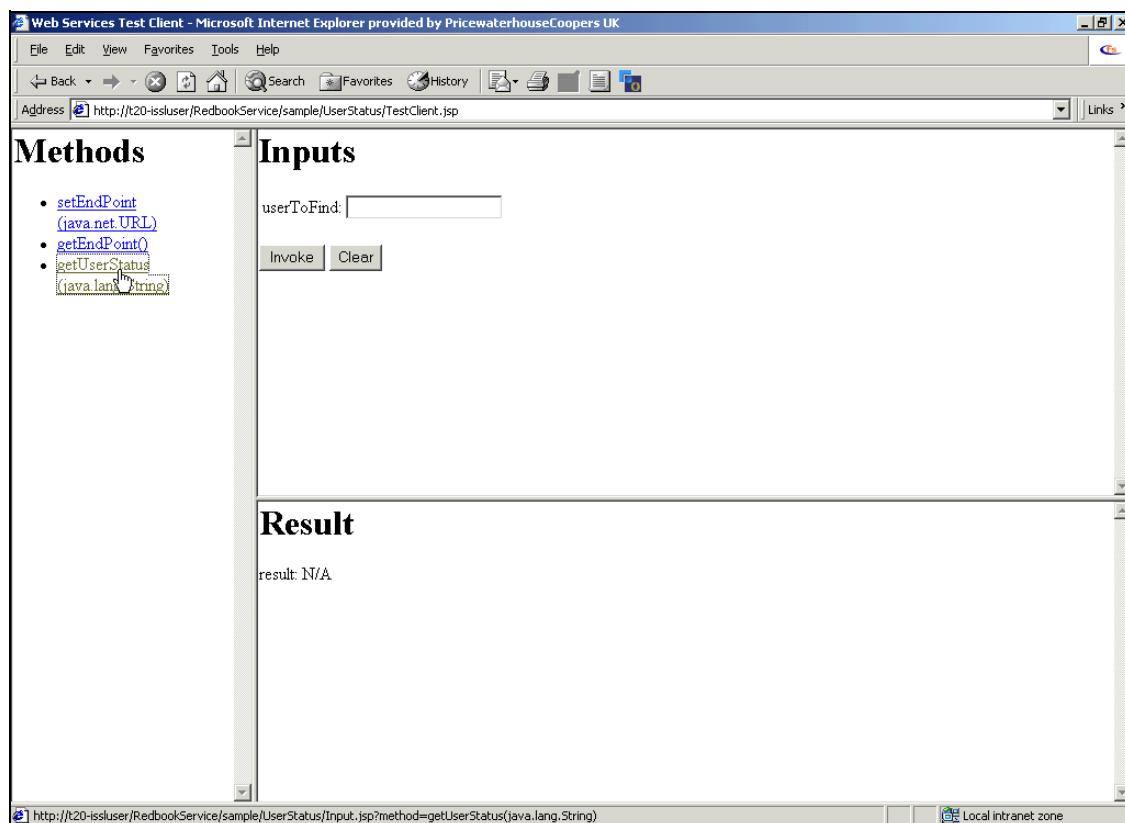


Figure 4-29 Accessing TestClient.jsp on WebSphere Application Server

2. Test the service by entering the name of a valid Sametime user. You will receive their current status in the Result.jsp frame (Figure 4-30 on page 124).

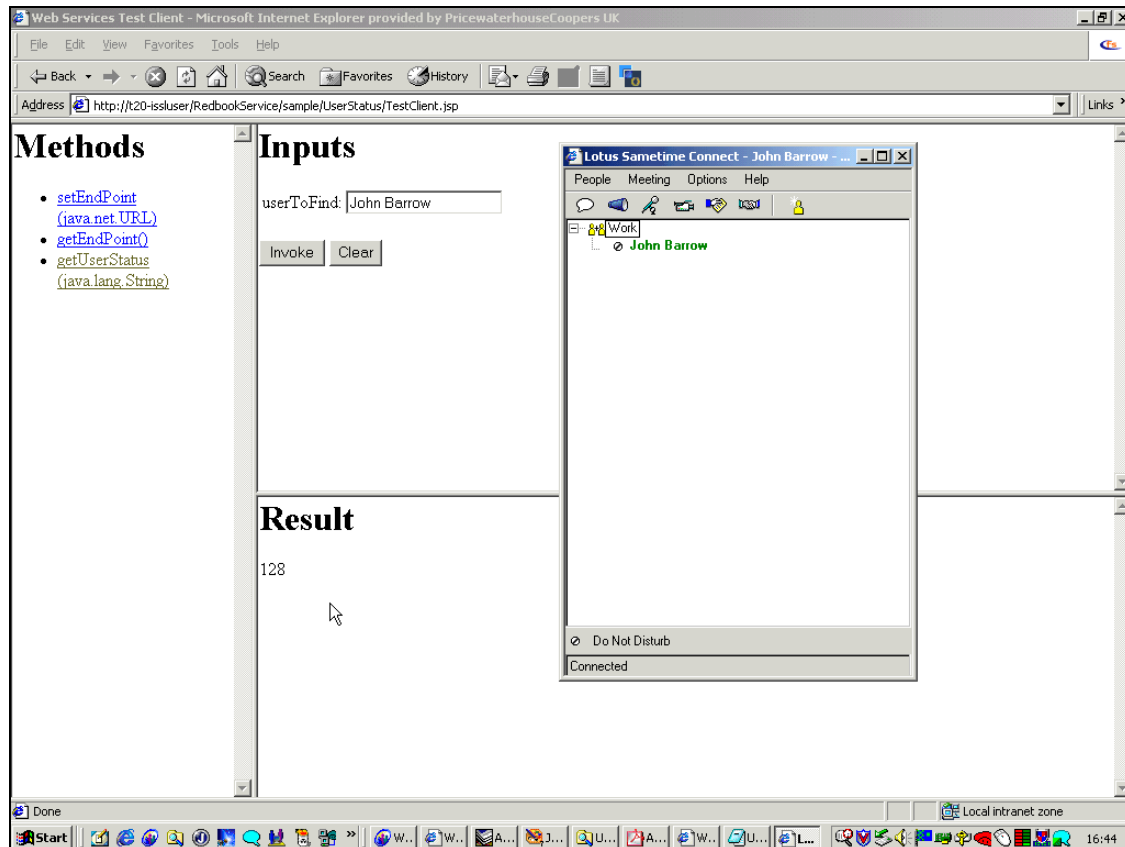


Figure 4-30 Testing the service with a valid Sametime user

The next logical step would be to create a friendlier user interface for this Web service. We could remove the frames and update the page with the Invoke button. We could create or enhance an end-user business application that would use this Web service to determine a user's Sametime status.

In the following chapters, we show how Web services can be built to expose many different facets of Sametime functionality, and how they can be incorporated into many different types of applications.

Important: This chapter has not attempted to provide a serious discussion on Web services, nor the intricacies of using WebSphere Studio Application Developer's Web services wizards. We have attempted to show how easy it is to create a Web service that exposes Sametime functionality. For a much more in-depth study of Web services and using WebSphere Studio Application Developer to create them, we encourage you to read *WebSphere Version 5 Web Services Handbook*, SG24-6891.



Chat Logging/DDA Toolkit

This chapter discusses the chat logging Service Provider Interface (SPI) and the Database and Directory Assistance Toolkit (DDA). It examines the reasons for chat recording, discusses considerations for implementing chat recording, discusses a sample available in the DDA Toolkit, and explains options for customization of the sample. The Sametime DDA Toolkit documentation provides a breakdown of the actual code in the sample and provides more direction for developers wishing to customize the example using C++.

5.1 Overview of the Chat Logging SPI and DDA Toolkit

In 2001, IBM introduced the Sametime Chat Recording Service Provider Interface (SPI), allowing developers to create Win32 DLL files to work in conjunction with a Win32 service to capture transcripts of chats as the sessions concluded. Corporate developers used the SPI to create custom versions of a chat recorder facility, saving transcripts to disk, text files, relational, or Notes databases as the company's requirements dictated. Commercial developers created a small number of products to do the same, with more configuration options to serve a larger market.

With the release of Sametime 3, the SPI was subsumed into the Database and Directory Assistance Toolkit (DDA). This chapter discusses the reasons for chat recording, considerations for implementing chat recording, discusses a sample available in the DDA Toolkit, and explains options for customization of the sample. The Sametime DDA Toolkit documentation provides a breakdown of the actual code in the sample and provides more direction for developers wishing to customize the example using C++.

Why would we discuss chat recording in a Sametime Web development redbook? The two subjects seem pretty distant; is not chat recording an admin task? Let us start by examining the reasons for chat recording.

There are four primary reasons an enterprise would want to record the instant message sessions occurring on a Sametime server:

- ▶ Regulatory compliance
- ▶ Collaborative commerce rules and workflow
- ▶ Corporate and public sector governance
- ▶ Appropriate use review

5.1.1 Regulatory compliance

In the wake of corporate scandals, document shredding, and collapsed companies, 2003 was the year for new and extended document archiving regulations in the United States. For years companies have been required by various U.S. government agencies to retain paper records, and more recently e-mail records, for specified periods of time. In March, the New York Stock Exchange (NYSE) strongly recommended to its member companies that instant messages be treated the same as e-mail records for retention purposes. In May, the Securities and Exchange Commission (SEC) implemented new regulations detailing the types of instant message conversations that by law must be retained and archived. In July, the National Association of Securities Dealers (NASD),

which runs the NASDAQ exchange, echoed the SEC by issuing strong recommendations to its own membership of securities dealers.

Clearly, the intent of these regulations and recommendations from government and financial industry bodies is twofold: to protect the interests of investors by holding investment professionals to the same standards in IM communications as in written communications, and to provide an audit trail for disputed transactions. Chat recording is now a requirement for certain types of IM traffic that may be enabled using many of the techniques described in this redbook. For example, the investor with secured access to a Sametime Links-enabled Web site may communicate with their investment professional about many subjects covered by the archiving laws.

5.1.2 Collaborative commerce rules and workflow

As IM becomes more ubiquitous using the techniques described in this redbook, companies are implementing it to enable their customers, suppliers, and other business partners to seek instant assistance when making decisions or conducting a transaction. Consider the furniture company with four main departments, each with a Sametime Links-enabled area of the corporate Web site. Adding Sametime Links by itself, with the modifications described in Chapter 9, “Sametime Links” on page 219, is a major step forward in bringing the consumer (furniture buyer) closer to the seller (customer service rep), and increases the chances of making a sale online.

However, the information passed between the buyer and seller is quite valuable itself. Consider the idea that a record of the chats from the seller’s Web site might reveal patterns of questions, ideas for new styles, or information that could be placed on the Web site itself (if asked often enough). In addition, if metadata is captured during the session (the buyer’s e-mail address, for example), someone could follow up with the buyer much more quickly if the record of their session with the customer service rep were available. Using chat recording in combination with Domino, each department’s transcripts could be mailed to a department supervisor for immediate follow-up, closing the loop on the transaction.

5.1.3 Corporate and public sector governance

Even without the pressures of the SEC, NYSE, and NASD, many companies are taking a proactive approach to corporate governance and record keeping. With global companies announcing accounting irregularities on a monthly basis, and pressure from governments for more “corporate responsibility”, many boards and CEOs are in the process of implementing programs to make their company records more transparent and of high ethical standards. In addition, the

remaining major accounting firms and auditors are recommending that all electronic communications be auditable.

While less ominous than a barrage of government regulations, the move towards corporate responsibility has a tendency to trickle down the organization and has caused many IT organizations to implement new archiving rules, including maintaining IM transcripts. In the public sector, open meeting and so-called “sunshine” laws that require open records of local and state governments (in the United States) are also being extended to IM and e-mail messages, in effect making them public record. Of course, in order to be a public record, there must be a record kept (a transcript, in the case of IM).

5.1.4 Appropriate use review

Last but not least, we use a quite sanitary term for Big Brother. It is true that in many organizations, IM traffic and e-mail are being monitored for the purpose of identifying inappropriate use of company resources. Without passing judgment on the practices, it is fair to say that companies have varying levels of rights (and responsibilities) to ensure that the use of instant messaging is legal, appropriate, and business oriented. Privacy laws in some regions of the world provide more protection for individuals, however, the pressures of regulations and competition also cause companies to have a need to protect their own interests, especially in matters that (were records not kept) would cause them and their shareholders financial harm.

Now that we have outlined some of the reasons for IM chat recording with Sametime, let us discuss how it is implemented.

5.2 Developer considerations

Sametime chat logging can be enabled on two types of Sametime text messages:

- ▶ One-to-one instant messages
- ▶ N-way chats and meetings (IM component)

The Sametime server implements two methods for processing chats. Basic, one-to-one IMs are connected by the servers, and at that point the servers act simply as routers for encrypted (by default) messages. N-way chats implement a mechanism called a *Place*, (which is described in detail in 8.1.1, “What are Places and why use them?” on page 184) to exchange messages among multiple participants. Either type of message traffic can be intercepted by a chat recorder built using the Chat Recorder SPI. Although the Sametime Connect client allows users to chat with AOL Instant Messenger users (AIM), you cannot

build a chat logger to capture these with the Chat Logging SPI. This is due to the fact that the service is server-based, and AIM chats are client-based.

Note: The Chat Recorder SPI only allows the server to capture chat traffic, and not entire meetings. There is a separate facility for recording meetings in the Sametime meeting setup options.

5.2.1 Modes

There are two modes of chat logging available: strict and relaxed. Strict logging requires that a message be successfully logged in order to continue the chat session. If an error is encountered, the chat session is terminated. Relaxed logging logs all the messages it encounters, with or without errors, always allowing chat to continue even if logging fails. All servers in a multiple server environment must be set to run in the same mode.

5.2.2 Distributed environments

As multiple Sametime servers can coexist and work together in an organization, chat logging follows specific rules on which server is chosen to intercept the chat and record it. For an N-way chat or meeting, the chat is logged on the server handling the Place. This could be one of many servers. For a simple instant message session, the chats are logged on the home server of the user accepting the invitation to chat (recipient).

In a distributed environment, all Sametime servers should be configured to either have chat logging enabled or not. If this is not the case, some chats will be logged and some will not, and the appearance of transcripts will be arbitrary.

5.2.3 Synchronous and asynchronous implementations

A chat logging DLL can be implemented asynchronously (message is sent regardless of its logged state) or synchronously (message is not sent until it is logged successfully). Sametime provides the Chat Logging SPI sample for synchronous implementation. You can develop your own chat logging DLL with asynchronous implementation.

Note: Chats logged in synchronous mode slow performance of the Sametime instant messages, because chats must be successfully logged before being forwarded to the recipient.

More developer considerations can be found in the *Sametime 3.0 Directory and Database Access Toolkit Developer's Guide* (see 2.2.1, "Sametime Software

Development Kit (SDK) documentation” on page 22 for information on finding this guide).

5.3 Toolkit examples

The Sametime 3.0 Directory and Database Access Toolkit provides example files to help developers build a chat logging DLL. Specifically, there is a Chat Logging SPI Example, with a full set of constants, error messages, and information on modifying `sametime.ini`. There is also a Chat Logging SPI Template, from which developers should develop new implementations of the SPI. Finally, there are Chat Logging SPI header files with definitions, syntax information, variables, constants, return values, and so on.

The toolkit, however, does not contain a fully built version of a chat logging DLL; rather, it has the files necessary to modify and build one. The skills required to do this include C++ skills and an understanding of the Notes C API, assuming that you will be recording to a Notes database. If you will be recording to a different repository other than Notes, you will also need to know how to use that target repository (Notes, SQL, disk file, and so on).

The DDA Toolkit is, like all the other kits, shipped as part of the Sametime "components" distribution. All of the files can be found in the file named `st30ddatk.zip`. The sub-directories that you will need for developing a chat logging plug-in are named "inc" (includes), and "templates" (skeleton code files). There are files in these folders for both chat logging and for authentication plug-ins, but we will only be concerned with chat logging in this section.

The kit also includes a "samples" folder, with code that you can look at. Unfortunately, that code is not actually buildable, because it requires some `#include` files that are not part of the kit. Our recommendation is to start with the code in templates, and ignore (or just skim) the sample code.

The template DDA code in Version 3.0 that we care about consists of a few header files, a single C++ source file, and a `.def` file:

```
inc\common\stDdaApiDefs.h
inc\chatlogging\stDdaC1Api.h
inc\chatlogging\stDdaC1Codes.h
templates\chatlogging\stDdaC1Api.cpp
templates\chatlogging\stDdaC1.def
```

Example 5-1 on page 133 shows the template C++ code that implements the framework for the chat logging plug-in.

Example 5-1 stDdaClApi.cpp

```
#include "stDdaClCodes.h"
#include "stDdaClCodes.h"

#include <stdio.h>
#include <string.h>
#include <stDdaClApi.h>

#define VP_CL_VERSION"3.16.0.4"

char *globalRetMsg = 0;
StDdaRetVal *globalRetCode = 0;

// *****
// DESCRIPTION: stDdaClInit
// *****
int ST_DDA_API
stDdaClInit( /*in/out*/ int* prVersion,
             /*in*/ int initializedOutside,
             /*out*/ int* initializedInside,
             /*out*/ char* dirType,
             /*out*/ char* libVersion,
             /*out*/ StDdaRetVal* appRetCode,
             /*out*/ char* appRetMsg)
{
    if (dirType && libVersion) {
        strcpy(dirType, ST_DDA_API_DUMMY_TYPE);
        strcpy(libVersion, VP_CL_VERSION);
    }
    // init debug mechanism

    globalRetCode = appRetCode;
    globalRetMsg = appRetMsg;
    *globalRetCode = ST_DDA_OK;

    if (*prVersion < ST_DDA_CL_LIB_VERSION)
    {
        *globalRetCode = ST_DDA_ERROR;
        sprintf(globalRetMsg, "Version - %d is not supported", *prVersion);
        return ST_DDA_API_VERSION_MISMATCH;
    }
    *prVersion = ST_DDA_CL_LIB_VERSION;

    (*initializedInside) |= initializedOutside;
    return ST_DDA_API_OK;
} // end stDdaClInit
```

```

// *****
// DESCRIPTION: stDdaClTerminate
// *****
void ST_DDA_API
stDdaClTerminate()
{

} // end stDdaClTerminate


// *****
// DESCRIPTION: stDdaClTimerEvent
// *****
void ST_DDA_API
stDdaClTimerEvent(void)
{
} // end stDdaClTimerEvent


int ST_DDA_API
stDdaClSessionStarted(/*in*/ const char* sessionId)
{
    return ST_DDA_API_OK;
}


// *****
// DESCRIPTION: stDdaClSessionStartedByOrgName
// *****
int ST_DDA_API
stDdaClSessionStartedByOrgName(/*in*/ const char* sessionId,
                               /*in*/ const char* organization)
{
    return ST_DDA_API_OK;
} // end stDdaClSessionStartedByOrgName


// *****
// DESCRIPTION: stDdaSessionEnded
// *****
int ST_DDA_API stDdaClSessionEnded(const char* sessionId)
{
    return ST_DDA_API_OK;
} // end stDdaSessionEnded

```



```

// *****
// DESCRIPTION: stDdaClUserJoinedSession
// *****
int ST_DDA_API stDdaClJoiningSession(/*in*/ const char* sessionId,
                                     /*in*/ const StDdaClEntity *entity,
                                     /*in*/ const StDdaClEntity *scope)
{
    return ST_DDA_API_OK;
} // end stDdaClUserJoinedSession


// *****
// DESCRIPTION: stDdaClUserLeftSession
// *****
int ST_DDA_API stDdaClLeavingSession(/*in*/ const char* sessionId,
                                     /*in*/ const StDdaClEntity *entity)
{
    return ST_DDA_API_OK;
} // end stDdaClUserLeftSession


// *****
// DESCRIPTION: stDdaSessionMsg
// *****
int ST_DDA_API
stDdaClSessionMsg(/*in*/ const char* sessionId,
                  /*in*/ const StDdaClEntity *sender,
                  /*in*/ unsigned long msgLen,
                  /*in*/ const char *msg,
                  /*in*/ const StDdaClEntity *receiver)
{
    return ST_DDA_API_OK;
} // end stDdaSessionMsg

```

The first thing to notice about this set of code is that the chat logging facility works (as you might expect) entirely on callbacks from the Sametime server. This file implements all of the required entry points that the server will invoke for various event occurrences. Note that the chat logging SPI is documented in the toolkit file `stddatkddevguide.pdf`:

- ▶ `stDdaClInit`: Called one time when the server loads the chat logging DLL. This is where you do all of your one-time setup, such as initializing the Notes API.
- ▶ `stDdaClTerminate`: Called one time when the Sametime server is about to shut down. Anything that you allocated or initialized in the `Init` call would be deallocated or terminated in this call.

- ▶ `stDdaCITimerEvent`: This entry point is documented as being invoked by the server approximately every minute. In Versions 2.5 and 3.0, however, it appears to never be invoked at all. The idea of this call is a good one: it gives you a chance to check for changed settings and/or other outside events on a periodic basis. Since it does not work, however, if you need to do periodic work, your best bet is to launch a Windows background thread that wakes up every so often to perform the housekeeping functions you need.
- ▶ `stDdaCISessionStarted`: This entry point is called whenever a new user session begins. You get passed the session ID, which is a string.
- ▶ `stDdaCISessionStartedByOrgName`: This entry point is not documented.
- ▶ `stDdaSessionEnded`: This is your notification that a session has ended. If you are logging transcripts by session (which is what most people would normally do), then this is your signal that you can close the transcript.
- ▶ `stDdaCIUserJoinedSession`: This routine is called whenever a user joins a session. You get an instance of the `StDdaCIEntity` struct (defined in the `StDdaCIApi.h` header file), which contains a name (string) and an "entity type". The entity type (an enum defined in the same file) could be a person, but it could also be a Sametime activity or a section).
- ▶ `stDdaCIUserLeftSession`: Your notification that someone (or something) has left the session.
- ▶ `stDdaSessionMsg`: This is the real heart of the system. It is invoked at this entry point for every message sent through the server. Arguments include the session ID, the sender, the receiver and the message itself. Note that Sametime makes no guarantees about the character set used to encode the message: it will be in whatever character set the client software sent to the server. This is inconvenient, to say the least, but you will just have to make some assumptions.

5.4 Customizing and building a Chat Logger

The way you plug a chat logging module into the Sametime server is to build a DLL named `StChatLog.dll`, which contains at least the required set of exported entry points. You replace the dummy `StChatLog.dll` file that ships with Sametime in the Sametime/Domino executable directory. Your DLL gets loaded automatically when the Sametime server starts up, assuming that chat logging is an enabled activity in the server configuration settings.

One common customization that you might want to make is to allow your logging code to use the Notes C API. The `stDdaCIInit()` call provides a mechanism by which you can find out if the server has already done this for you or not.

Example 5-2 shows a modified `stDdaClInit()` that takes care of this step.

Example 5-2 Modified `stDdaClInit()` code to handle Notes C API Initialization

```
int ST_DDA_API
stDdaClInit( /*in/out*/ int* prVersion,
             /*in*/    int initializedOutside,
             /*out*/   int* initializedInside,
             /*out*/   char* dirType,
             /*out*/   char* libVersion,
             /*out*/   StDdaRetVal* appRetCode,
             /*out*/   char* appRetMsg)
{
    if (dirType && libVersion)
    {
        strcpy(dirType, "Looseleaf Chat Recorder");
        strcpy(libVersion, VP_CL_VERSION);
    }

    globalRetCode = appRetCode;
    globalRetMsg = appRetMsg;
    *globalRetCode = ST_DDA_OK;

    if (*prVersion < ST_DDA_CL_LIB_VERSION)
    {
        /*
         dbg.print("The recived version - %d not suppoted, black support only
version %d and up",
                *prVersion,
                ST_DDA_CL_LIB_VERSION);
        */
        *globalRetCode = ST_DDA_ERROR;
        sprintf(globalRetMsg, "Version - %d is not supported",
                *prVersion);
        return ST_DDA_API_VERSION_MISMATCH;
    }
    *prVersion = ST_DDA_CL_LIB_VERSION;

    // init notes, if necessary
    if (!(initializedOutside & ST_DDA_API_NOTES))
        RecorderInst = InitNotes();

    (*initializedInside) |= (initializedOutside | ST_DDA_API_NOTES);
    globalInitMask = *initializedInside;
    return ST_DDA_API_OK;
} // end stDdaClInit
```

Note particularly the highlighted code at the end. We check the initializedOutside input parameter to see if the server has already done this task for us. The #define ST_DDA_API_NOTES can be found in the stDdaApiDefs.h header. We call a local routine that simply calls NotesInitExtended(). The initializedInside output parameter is always a copy of the initializedOutside flags plus whatever we initialize ourselves (so that the server knows not to init Notes again). We keep a copy in a global variable so that we will remember to terminate Notes later.

The correspondingly modified stDdaCITerminate routine code is shown in Example 5-3.

Example 5-3 Modified stDdaCITerminate() code to Handle Notes API termination

```
void ST_DDA_API
stDdaCITerminate()
{
    if (globalInitMask & ST_DDA_API_NOTES)
        TermNotes(RecorderInst);
    globalInitMask &= ~(ST_DDA_API_NOTES);
} // end stDdaCITerminate
```

The local TermNotes() routine simply calls NotesTerm(), and does whatever other global cleanup our logger requires.

Note that neither the Init nor the Term routines requires synchronization to protect global variables (such as the globalInitMask variable in Example 5-2 on page 137 and Example 5-3). These routines are guaranteed to be called on a single thread, one at DLL load time, and one at server shutdown time. In fact, the entire Sametime chat logging SPI is documented as being single-threaded, which means that there is no need to write code defending against concurrent access to global state.

If your logger does substantial work on server invocations, however, the fact that the calls are synchronous from the server to the logger DLL could cause the server to slow unacceptably. The best way to work around this would be to have your callback routines simply take the given input parameters and copy them to an in-memory queue of some kind, and then return right away. You would then have one or more background threads that picked "messages" off the end of the queue and processed them.

Finally, to activate the created chat logging DLL, start and stop the Sametime server on which the DLL is installed. If the Sametime community contains more than one Sametime server, start and stop all servers in the community.



Sametime and workflow

This chapter discusses how Sametime can be used within a workflow application context. We show how you can connect your application to Sametime and leverage the functionality that it offers.

We discuss a number of different scenarios where you may wish to incorporate Sametime into existing business processes. We describe in detail how to enable a Domino workflow application with awareness and instant messaging capability.

We take this idea a step further and discuss how to utilize Sametime Web services in applications. Any application capable of acting as a Web services client can leverage the power of Sametime. We show a Web service that acts as an announcement service, allowing applications to send instant announcements to selected online users.

By using the functionality provided by Sametime Links (STLinks), we expand on the announcement service concept. Using the STLinks announcement dialog allows the service to send rich text, greatly increasing the value of your applications.

6.1 Using Sametime within a workflow

In Chapter 3, “Sametime Bots” on page 49, we showed examples of how a user can initiate a conversation with an application using Sametime as the interface. When you integrate Sametime into workflow applications, the reverse also becomes possible. Applications can become aware of users’ online availability and can initiate conversations with them.

Being aware of a user’s Sametime state is a very powerful feature. It lets you know whether or not that user is available and able to respond to a message from you. With e-mail and other types of messaging, you often send information with no real idea of whether or not there is somebody currently at the other end to read it. You may get a quick response, but you also may not. This immediacy of response may not be a big deal to you, but it may be critical. The ability to know whether or not a specific person is currently available can be an important component of a business-critical application.

Applications that use Sametime awareness can make decisions on the best medium to use when contacting a user. For example, an application could proactively send messages to online users, potentially speeding up the rate at which a particular business process moved. Alternatively, the application could use conventional e-mail to notify users not currently active in the Sametime community.

6.2 The scenario

Workflow applications are prime candidates to take advantage of the awareness and messaging capabilities of Sametime. They typically involve a business process moving through some form of approval routing. Many workflow systems use e-mail to notify users at each point in the approval process. These e-mails frequently contain links back to the workflow application to allow users to perform the relevant action.

In many cases, such a scenario is perfectly acceptable. But on occasion, the immediacy and directness of instant messaging could make a critical difference. There are numerous applications that require a user to be notified of some event, for example, a change in state or an exception to an anticipated result.

For example, consider an application that monitored the availability of a mission-critical server. The application would alert administrators if the server crashed or reached a certain performance threshold. This alert could be automatically delivered to cell phones, pagers, or via e-mail. But if the application was Sametime-enabled and the administrators were online and active, they could be notified instantly via instant message or announcement. The instant message

could provide summary information generated by the application, which could provide further information in the form of a chat if requested by the administrator. The application would have the ability to determine the best method of contacting the people who need to be informed. If they are not active in Sametime, the application could select the next best alternative, such as text pager or e-mail.

6.3 The AnnouncementSender application

A workflow application developed using Lotus Domino is a fairly common entity. Workflow is one of Domino's key strengths, utilizing the built-in support for e-mail and security. Documents can move through a defined series of approval stages, with e-mail notification at each step, usually including a link back to the document awaiting approval. These applications can be easily Web-enabled to allow access from Web browsers as well as Notes clients.

In this example, we add Sametime functionality to our workflow database. Rather than using e-mail as our notification medium, if the user is online and active, we use Sametime's Announcement Service. This feature was new with Sametime 3.0 and delivers the message text in a dialog box without the ability for the user to respond. This is very useful, as in this particular case we do not want the user to carry on a conversation with our announcement application. If the user is not active in Sametime, we can fall back to the old method and send an e-mail.

6.3.1 The AnnouncementSender agent

The AnnouncementSender agent is a Java agent that uses the Java Client Toolkit. It takes a user name as a parameter, checks their online status, and, if active, sends them an announcement. The code for this agent is very similar to that in 4.3.1, "The UserStatus application" on page 86, so we only discuss the additions here.

We start by importing the Sametime-specific packages (Example 6-1). The only new package in this application is the Announcement Service.

Example 6-1 Package imports

```
import lotus.domino.*;
import com.lotus.sametime.announcement.*;
import com.lotus.sametime.awareness.*;
import com.lotus.sametime.community.*;
import com.lotus.sametime.core.comparch.*;
import com.lotus.sametime.core.constants.*;
import com.lotus.sametime.core.types.*;
import com.lotus.sametime.lookup.*;
```

The class is a Domino java agent, so it extends the AgentBase class. It also implements three Sametime listener interfaces. We have a new variable for our AnnouncementService object, and a new String variable for the actual text of our announcement (see Example 6-2).

Example 6-2 Class and variable declarations

```
public class JavaAgent extends AgentBase implements LoginListener,
ResolveListener, StatusListener
{
    private STSession m_session;
    private CommunityService m_communityService;
    private LookupService m_lookupService;
    private AnnouncementService m_announcementService;
    private AwarenessService m_awarenessService;
    private Resolver m_resolver;
    private WatchList m_watchList;
    private STUser m_requestedUser;

    private String m_strUserName;
    private String m_strAnnouncement;
    private short m_shAnnouncementStatus;
    private boolean m_bSetup;
    private boolean m_bResolved;
    private boolean m_bFinished;
```

In the agent's NotesMain() method (Example 6-3), we call the sendAnnouncement() method, passing in the user name and announcement text as parameters. The method returns a short signifying whether or not an announcement was sent. The announcement is not sent unless the user is online and active.

Example 6-3 NotesMain() method

```
public void NotesMain()
{
    try
    {
        Session session = getSession();
        AgentContext agentContext = session.getAgentContext();

        // (Your code goes here)
        short shAnnouncementStatus = sendAnnouncement("John Barrow", "Hello
World");
        System.out.println("Status of announcement for user John Barrow: " +
shAnnouncementStatus);
    }
    catch(Exception e)
    {
```



```

        e.printStackTrace();
    }
}

```

The `sendAnnouncement()` method attempts to log in to the Sametime server, and if successful, calls the `resolve()` method.

In the `loggedIn()` event (Example 6-4), we get a handle on the `AnnouncementService` object.

Example 6-4 The `loggedIn()` event

```

public void loggedIn(LoginEvent event)
{
    System.out.println("loggedIn()");
    m_resolver = m_lookupService.createResolver(false, false, true, false);
    m_awarenessService = (AwarenessService)
m_session.getCompApi(AwarenessService.COMP_NAME);
    m_watchList = m_awarenessService.createWatchList();
    m_watchList.addStatusListener(this);
    m_announcementService = (AnnouncementService)
m_session.getCompApi(AnnouncementService.COMP_NAME);
    m_bSetup = true;
}

```

The `resolve()` method attempts to resolve the user name into a valid Sametime `STUser` object. If the resolve is successful, the resolved user is added to a `WatchList` object when the `resolved()` method is called. Adding the `STUser` object to the `WatchList` causes the `userStatusChanged()` method to be called (Example 6-5). At this point, we check to see if the user is active in Sametime and if so, send them the announcement.

Example 6-5 `userStatusChanged()` event

```

public void userStatusChanged(StatusEvent event)
{
    System.out.println("userStatusChanged()");
    STWatchedUser[] watchedUser = event.getWatchedUsers();
    short userStatus = watchedUser[0].getStatus().getStatusType();
    if(userStatus == 32)
    {
        STObject stObject[] = {m_requestedUser};
        System.out.println("sendAnnouncement");
        m_announcementService.sendAnnouncement(stObject, false,
m_strAnnouncement);
        m_shAnnouncementStatus = 0;
        m_bFinished = true;
    }
}

```

```

else
{
    m_shAnnouncementStatus = -1;
    m_bFinished = true;
}
}

```

To get the agent to compile successfully, STComm.jar must be on Notes' classpath. There are a couple of different ways to achieve this:

1. Import the jar file into the Java agent itself. Open the agent in Designer and click on the **Edit Project** button. Navigate to the location of the STComm.jar file and click the **Add/Replace file(s)** button (Figure 6-1).

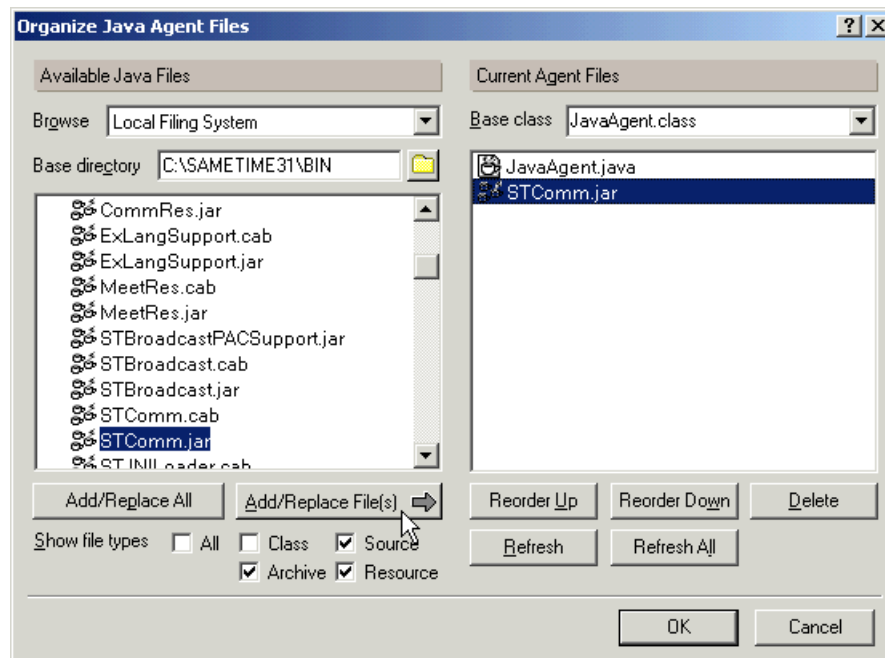


Figure 6-1 Adding toolkit jar file to agent project

2. Importing files into your agent project in the way discussed above makes your agent very inefficient. To improve performance, you can add these files to your classpath by directly specifying them in the Notes.ini file of your Notes client. If the agent is to run on the server or on a schedule, you will need to get your Domino administrator to modify the server's Notes.ini. In either case, you will need to add the following line and restart your client or server:
JavaUserClasses=C:\STComm.jar

You can test the agent by running it from inside Domino Designer®. Highlight the agent and select Actions → Run. You should see a Sametime announcement dialog, as shown in Figure 6-2.

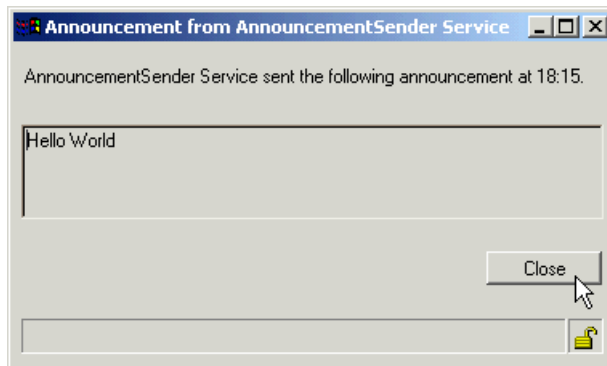


Figure 6-2 Sametime announcement dialog sent by agent

With a few extra lines of code, we can hook this agent into a real Notes workflow application. Figure 6-3 shows a Notes form that contains two fields and a button.

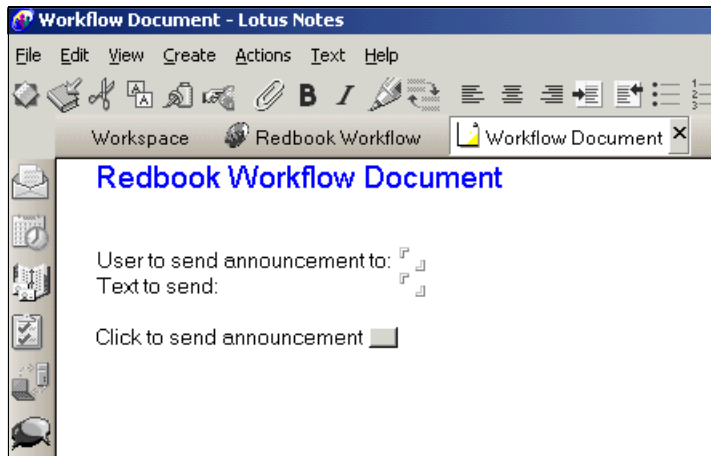


Figure 6-3 Notes workflow form

Example 6-6 on page 146 shows the LotusScript that is the event of the Click button.

Example 6-6 Click button event

```
Sub Click(Source As Button)

    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim agent As NotesAgent
    Dim ws As New NotesUIWorkspace
    Dim uidoc As NotesUIDocument
    Dim paramid As String

    Set uidoc = ws.CurrentDocument
    Set doc = uidoc.document
    Call doc.save(True, False)
    Call uidoc.save
    Call uidoc.close

    paramid = doc.Noteid
    Set db = session.CurrentDatabase
    Set agent = db.GetAgent("AnnouncementSender")
    Call agent.run(paramid)

End Sub
```

When the button is clicked, the document is saved and the AnnouncementSender agent is called. The agent is passed the Note ID of the document as a parameter to the run() method.

If we add the code shown in Example 6-7 to our AnnouncementSender agent, it can look up the workflow document and extract the user name and message text from it. It can then send the announcement containing the specified text to the end user.

Example 6-7 Modified NotesMain() method to include lookup code

```
public void NotesMain()
{
    try
    {
        Session session = getSession();
        AgentContext agentContext = session.getAgentContext();

        Database db = agentContext.getCurrentDatabase();
        Agent ag1 = agentContext.getCurrentAgent();
        String paramid = ag1.getParameterDocID();
        Document doc = db.getDocumentByID(paramid);
        String userName = doc.getItemValueString("username");
        String announcementText = doc.getItemValueString("announcement");
    }
}
```

```

        // (Your code goes here)
        short shAnnouncementStatus = sendAnnouncement(userName,
announcementText);
        System.out.println("Status of announcement for user " + userName + ":
" + shAnnouncementStatus);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

The agent can extract the Note ID parameter using the `getParameterDocID()` method of the Agent class.

If we now create a workflow document, we can specify the recipient and text of our announcement (see Figure 6-4 on page 148).

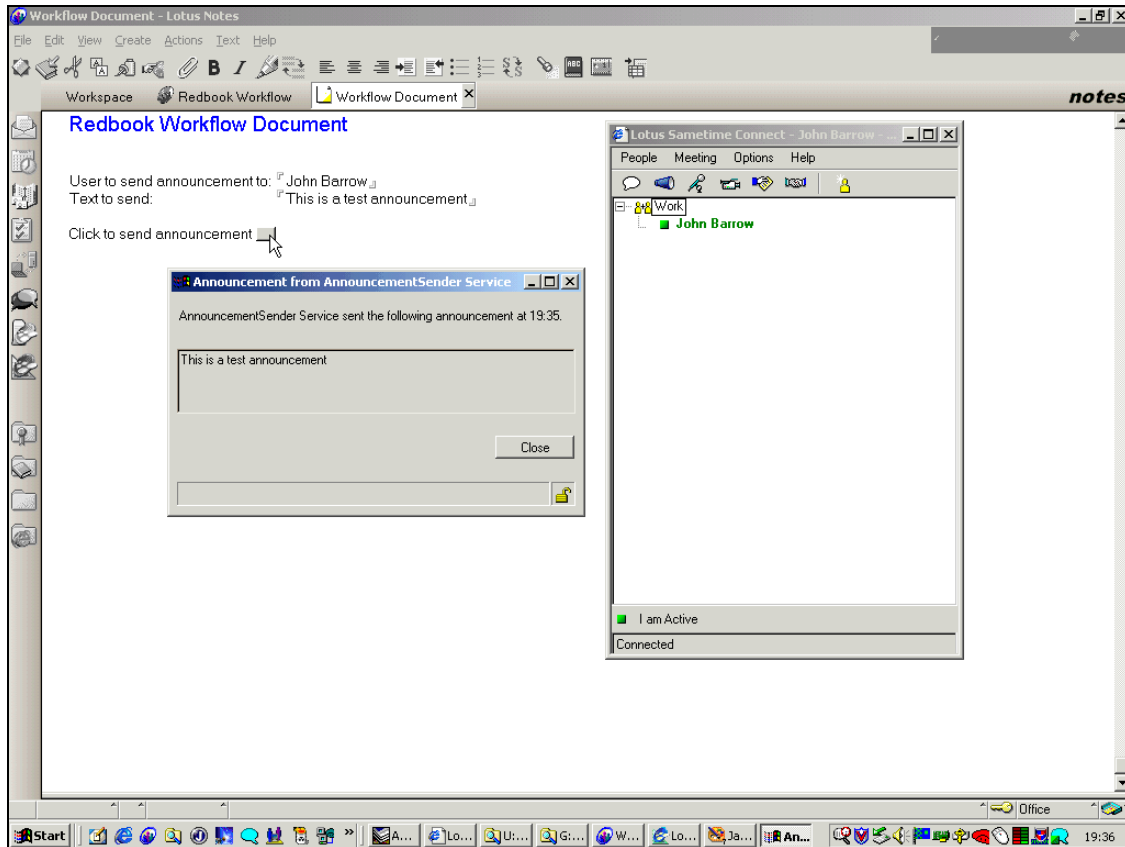


Figure 6-4 Announcement sent to active Sametime user

We could, of course, add further code to our agent to handle offline or non-active Sametime users by adding an e-mail sending routine that was called if the announcement could not be sent.

6.4 The AnnouncementSender Web service

The previous section showed how to embed Sametime functionality into a Notes workflow application. In this section, we turn the AnnouncementSender agent into a Web service and deploy it on WebSphere Application Server. This allows the announcement-sending functionality to be leveraged by many more applications throughout the enterprise.

The process for turning the AnnouncementSender agent into a Web service is identical to that described in 4.4, "Creating the UserStatus Web service" on

page 97. We can use WebSphere Studio Application Developer's Web services wizards to create a java proxy class and the associated WSDL files. Once deployed to WebSphere Application Server, we can use the TestClient.jsp file to test the service.

Tip: See Chapter 4, "Web services" on page 83 for a full description of creating and deploying Sametime Web services.

You can see from Figure 6-5 that the `sendAnnouncement()` method has been exposed as a Web service and takes two String parameters: user name and announcement.

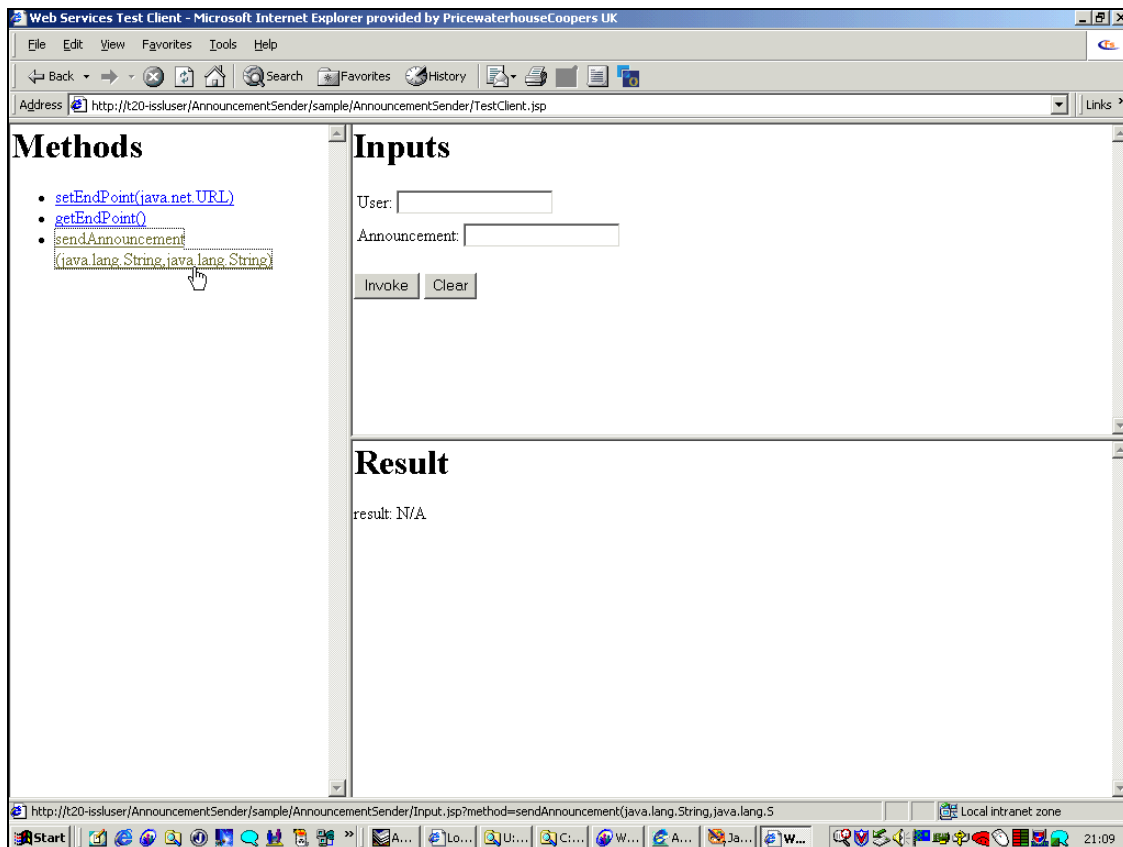


Figure 6-5 TestClient.jsp test harness for AnnouncementSender Web service

Figure 6-6 on page 150 shows the results if we use the service to contact an active Sametime user.

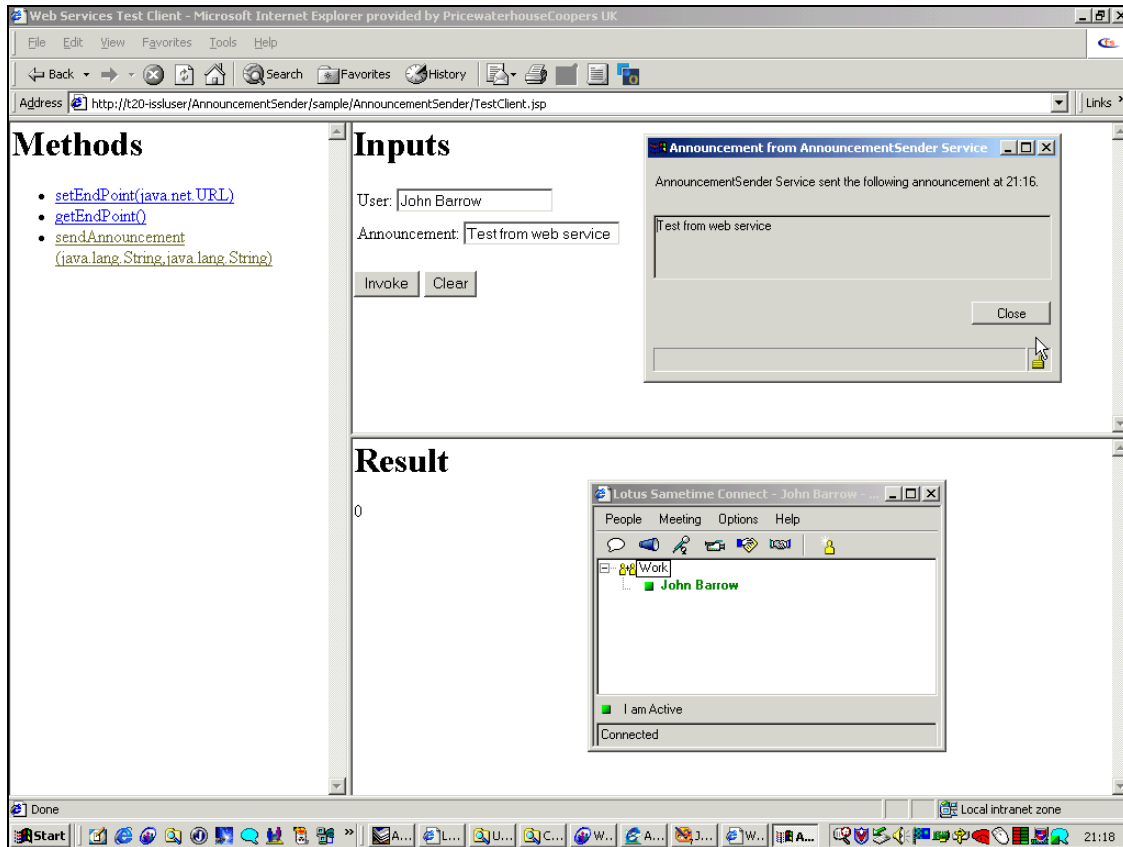


Figure 6-6 Announcement delivered using AnnouncementSender Web service

Now we have deployed the AnnouncementSender Web service, it makes sense to change our original workflow implementation to take advantage of it. We no longer need the AnnouncementSender Java agent with all its Sametime-specific code. We can replace it with a much more generic agent to call the AnnouncementSender Web service instead. Example 6-8 shows a Java agent that uses the Apache SOAP package (<http://ws.apache.org/soap/index.html>) to call the AnnouncementSender Web service.

Example 6-8 AnnouncementWebService Java agent

```
import lotus.domino.*;
import java.util.*;
import java.net.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
```



```

import org.apache.soap.transport.http.SOAPHTTPConnection;

public class JavaAgent extends AgentBase {

    public void NotesMain() {

        try {
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();

            Database db = agentContext.getCurrentDatabase();
            Agent ag1 = agentContext.getCurrentAgent();
            String paramid = ag1.getParameterDocID();
            Document doc = db.getDocumentByID(paramid);
            String userName = doc.getItemValueString("username");
            String announcementText = doc.getItemValueString("announcement");

            // (Your code goes here)
            short shAnnouncementStatus = sendAnnouncement(userName,
announcementText);
            System.out.println("Status of announcement for user " + userName + ":
" + shAnnouncementStatus);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public short sendAnnouncement (String User, String Announcement) throws
Exception
    {

        Call call = new Call ();

        // If you are behind a proxy, you will need to uncomment these four
lines....
        //SOAPHTTPConnection shc = new SOAPHTTPConnection ();
        //shc.setProxyHost(<proxyname>);
        //shc.setProxyPort(<proxyport>);
        //call.setSOAPTransport (shc);

        URL url=new
URL("http://t20-issluser:9080/AnnouncementSender/servlet/rpcrouter");

        // This service uses standard SOAP encoding
        String encodingStyleURI = Constants.NS_URI_SOAP_ENC;
        call.setEncodingStyleURI(encodingStyleURI);
    }
}

```

```

call.setTargetObjectURI ("urn:AnnouncementSender");
call.setMethodName ("sendAnnouncement");

// Create a vector for the parameters
Vector params = new Vector ();

Parameter strUserParam = new Parameter("User", String.class, User,
null);
params.addElement(strUserParam);
Parameter strMessageParam = new Parameter("Announcement", String.class,
Announcement, null);
params.addElement(strMessageParam);

call.setParams (params);

// Invoke the service ...
Response resp = call.invoke (url,"");

// ... and Evaluate the result
if (resp.generatedFault ())
{
    Fault fault = resp.getFault();
    throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
}
else
{
    // Successful result. Extract response parameter and return
    Parameter result = resp.getReturnValue ();

    return ((java.lang.Short)result.getValue()).shortValue();
}
}
}

```

For this agent to compile, you will need the 'standard' Web services jar files on your classpath: soap.jar, xerces.jar, activation.jar, and mail.jar.

Tip: For more information on using Web services with Domino, see *Domino Web Service Application Development for the IBM @server iSeries Server*, SG24-6862.

You will notice how much smaller this agent is compared to the AnnouncementSender agent. We can replace all of our Sametime-specific code with roughly a dozen lines of fairly generic Web services code instead.

We can now modify our Notes workflow form to use this new AnnouncementWebService agent instead. We can add a second button to our form that calls the new AnnouncementWebService agent. Figure 6-7 shows the result of using the new agent to send an announcement to an active Sametime user.

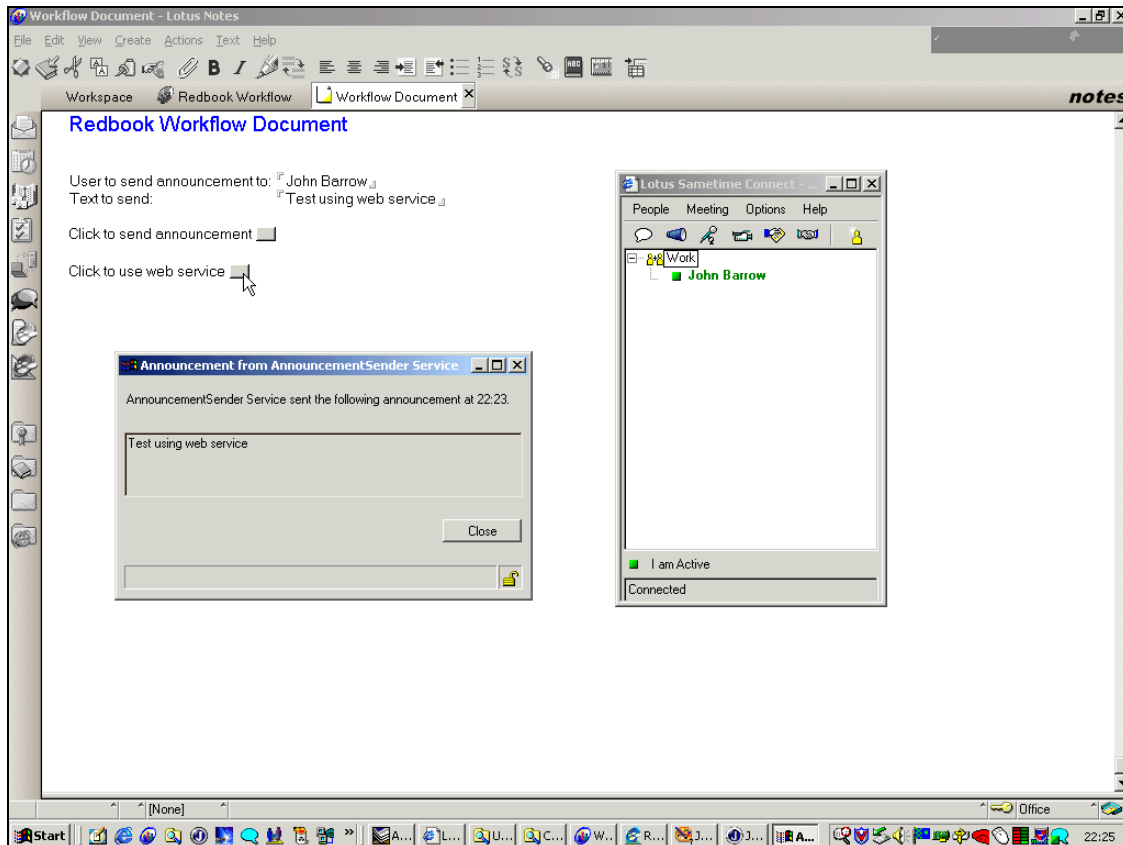


Figure 6-7 Using the AnnouncementWebService agent in the workflow application

6.5 Sending rich text announcements with STLinks

The messages delivered by Sametime to users of the Connect and Java Connect clients are currently limited to text. This is usually sufficient for the needs of a workflow application. However, if you use the toolkit Announcement Service in conjunction with the Sametime Links (STLinks) Web-based client, you have the ability to send rich text in the form of HTML. For example, Figure 6-8 on page 154 shows the results of using the AnnouncementWebService agent to

send an HTML tag to a user logged in to the Sametime community with STLinks.

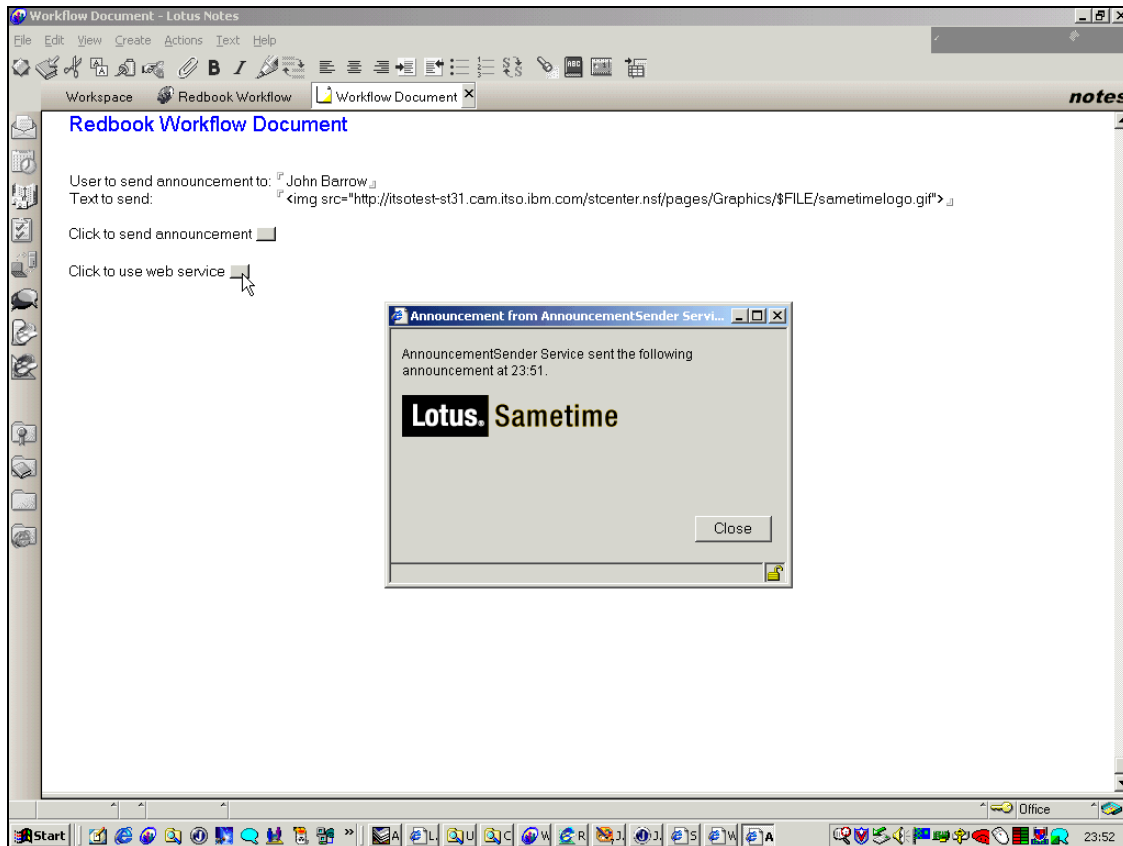


Figure 6-8 Using the AnnouncementWebService agent with STLinks

While the example of sending an tag may seem a little pointless in a workflow scenario, imagine how powerful it would be to be able to send an HTML <form> tag. The announcement window could then contain a form that required the end user to submit further information. The form could then be posted off to some other location for processing. This location could be a Java servlet on an application server, a Web service, or a CGI script on a Web server. As Figure 6-9 on page 155 shows, the possibilities are enormous.



Figure 6-9 Using the *AnnouncementWebService* agent to send an HTML form

Tip: The ability to send HTML in the STLinks announcement dialog is out-of-the-box functionality. If you wish to send HTML using the chat dialog, you will need to customize the HTML pages that make up the STLinks chat functionality. See Chapter 9, “Sametime Links” on page 219 for further details.

6.6 Integrating with Microsoft Excel

As was mentioned previously, any application that can make use of Web services can make use of the AnnouncementSender Web service to send messages to Sametime users. We have shown how its possible to interact with this Web service from Lotus Notes and Domino. What we do now is show how the Web service can be initiated by an event within Microsoft Excel.

You can see, in Figure 6-10 on page 156, an Excel Spreadsheet that has two simple forms. The first, in cells B2..C6, contains cells C3 and C4, which hold the Sametime user name to send the announcement to, and C5, which holds the message to be sent. Cell C6 contains the result code that is returned by the Web service when a message has been sent. The second form, in cells B11..13, has a cell that will send the announcement when the value in cell C12 goes above 100.

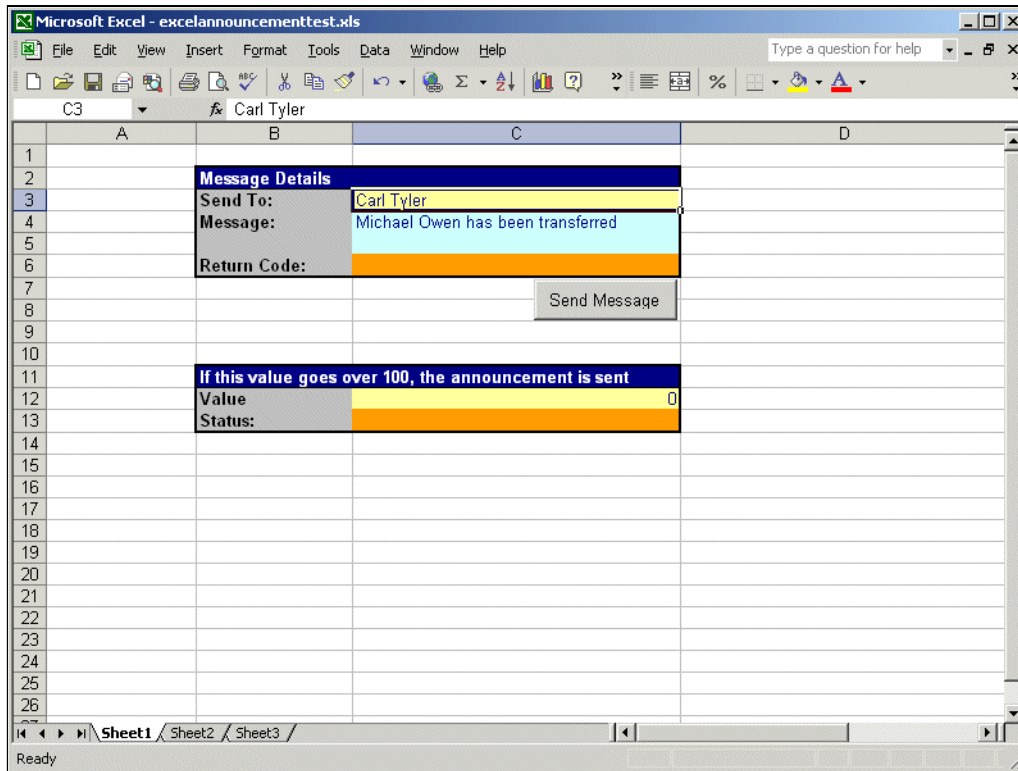


Figure 6-10 Microsoft Excel integrating with AnnouncementSender Web service

6.6.1 Enabling Microsoft Excel for Web services

Let us look more closely at the steps involved in enabling Microsoft Excel to interact with the AnnouncementSender Web service. The first step we need to take is to switch to the Excel development environment and include the Microsoft XML services. The Microsoft XML services we use are XML Services 2.0, which are installed with Microsoft Internet Explorer 5.0 and later.

1. Switch to the Microsoft Excel Visual Basic Editor by selecting **Tools** → **Macro Visual** → **Basic Editor**. This will present you with the Visual Basic Editor, as seen in Figure 6-11 on page 157.

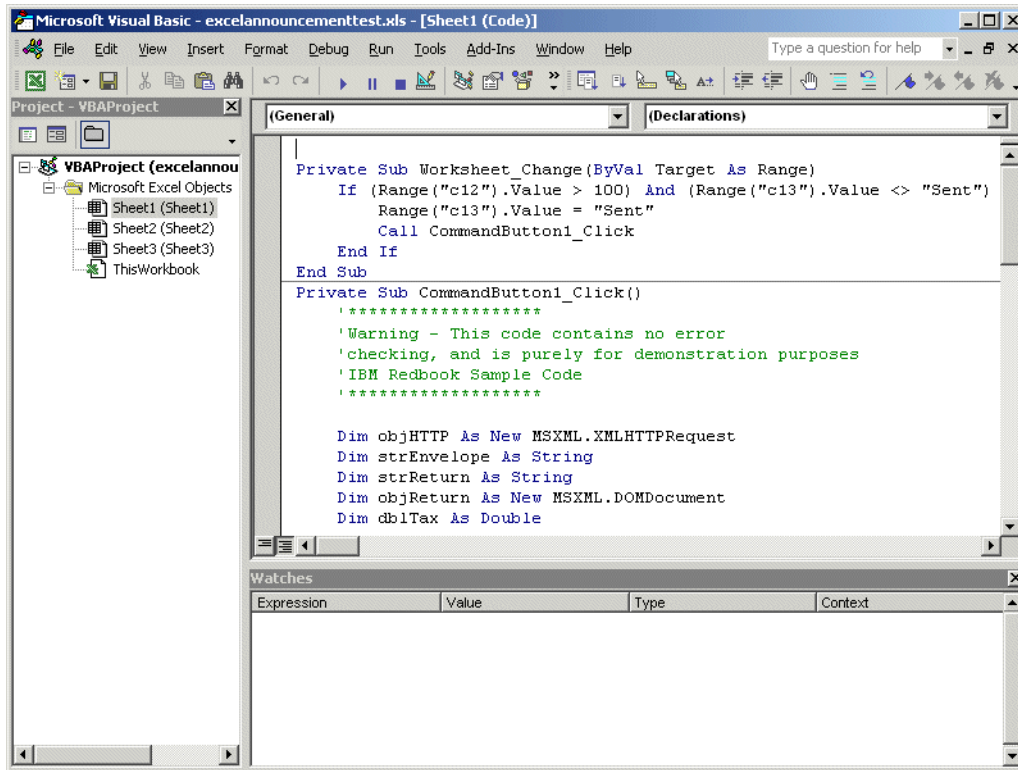


Figure 6-11 Microsoft Excel Visual Basic Editor

2. Now we need to enable Excel to reference the object library that is available with the Microsoft XML Services. This is done by selecting **Tools** → **References** and selecting **Microsoft XML, Version 2.0** from the list, and then clicking **OK**, as seen in Figure 6-12 on page 158.

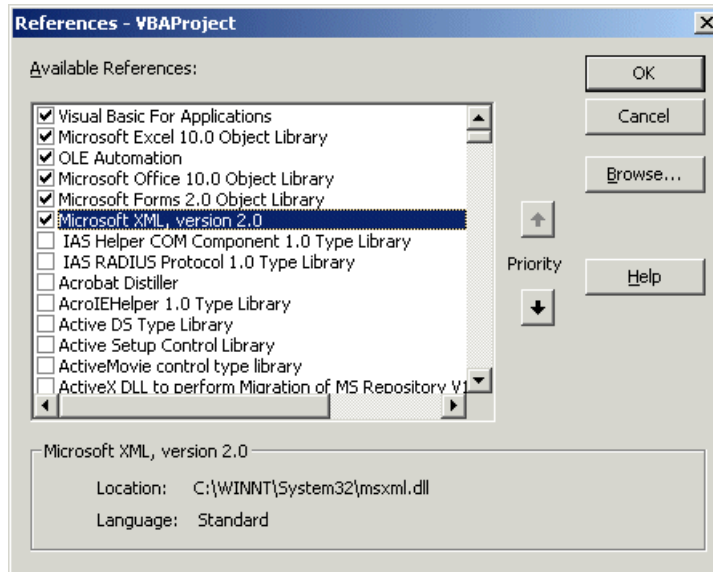


Figure 6-12 Enabling access to the Microsoft XML Object Library

3. The Microsoft XML services are now available to our Microsoft Excel application.

The function that calls the AnnouncementSender Web service via the Microsoft XML services is:

```
Private Sub CommandButton1_Click()
    '*****
    'Warning - This code contains no error
    'checking, and is purely for demonstration purposes
    'IBM Redbook Sample Code
    '*****

    Dim objHTTP As New MSXML.XMLHTTPRequest
    Dim strEnvelope As String
    Dim strReturn As String
    Dim objResults As New MSXML.DOMDocument ' Contains the results of the SOAP
Call
    Dim strQuery As String

    Dim MessageToSend As String
    Dim SendTo As String

    SendTo = Range("C3").Value
    MessageToSend = Range("C4").Value
```



```

'Create the SOAP Envelope
strEnvelope = _
"<?xml version='1.0' encoding='UTF-8'?>" & _
"<SOAP-ENV:Envelope
xmlns:SOAP-ENV=""http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd=""http://www.w3.org/2001/XMLSchema"">" & _
"<SOAP-ENV:Body>" & _
"<ns1:sendAnnouncement xmlns:ns1=""urn:AnnouncementSender""
SOAP-ENV:encodingStyle=""http://schemas.xmlsoap.org/soap/encoding/">" & _
"<User xsi:type=""xsd:string"">" & SendTo & "</User>" & _
"<Announcement xsi:type=""xsd:string"">" & MessageToSend &
"</Announcement>" & _
"</ns1:sendAnnouncement>" & _
"</SOAP-ENV:Body>" & _
"</SOAP-ENV:Envelope>"

'Set up the post to our WebSphere Server
objHTTP.Open "post",
"http://9.33.85.89:9080/AnnouncementSender/servlet/rpcrouter", False

'Define the Header as a standard SOAP/ XML header for the content-type
objHTTP.setRequestHeader "Content-Type", "text/xml"

'Set the header for the method being called
objHTTP.setRequestHeader "SOAPMethodName", "urn:AnnouncementSender"

'Make the SOAP call using the strEnvelope
objHTTP.send strEnvelope

'Get the results that are passed back in an envelope from the SOAP call
strReturn = objHTTP.responseText

'Put the Envelope that is returned into a cell on the spreadsheet
'If you wish to see the returned envelope un comment the following line
'Range("d3").Value = strReturn

'Load the return envelope into a DOM so we can pull out specific values
objResults.loadXML strReturn

'Pull out the return result state, -1 if the message failed, 0 if was sent
successfully
Range("C6").Value = objReturn.Text
End Sub

```

With this code, a SOAP envelope is created that is sent to the AnnouncementSender Web service. The lines:

```
SendTo = Range("C3").Value  
MessageToSend = Range("C4").Value
```

read the values that are in cells C3 and C4 into the variables SendTo and MessageToSend. The lines:

```
"<User xsi:type=""xsd:string"">" & SendTo & "</User>" & _  
"<Announcement xsi:type=""xsd:string"">" & MessageToSend & "</Announcement>" &  
_
```

are where the SendTo and MessageToSend variables are substituted into the SOAP message.

The remaining lines handle the interaction with the Microsoft XML Services, and handling the response. The last lines:

```
'Pull out the return result state, -1 if the message failed, 0 if was sent  
successfully  
Range("C6").Value = objReturn.Text
```

read back the result state from the Web service and places the value in cell C6. If the AnnouncementSender was successful, it returns 0, and if it failed, it returns -1.

The code within this example is activated when the user clicks on the **Send Message** button.

This code can easily be taken and extended into a workflow-type application. Within this example, we have enabled cell C12 to issue an announcement if the value should go above 100. The lines:

```
Private Sub Worksheet_Change(ByVal Target As Range)  
    'Check to see if cell c12 is above 100 if it is  
    'and cell C13 is not Sent then we will send the message  
    If (Range("c12").Value > 100) And (Range("c13").Value <> "Sent") Then  
        'Put the value sent into the cell C13  
        Range("c13").Value = "Sent"  
        'Call the function to send the announcement  
        Call CommandButton1_Click  
    End If  
End Sub
```

perform this function. The Function Worksheet_Change is called by Microsoft Excel everytime there is a change on the Worksheet. When this function is called, we check to see if the value in cell C12 is above 100. If it is, and the value in C13 is not set to "Sent", we can process the announcement. The first thing we do when this happens is to set the value of C13 to sent. This stops the AnnouncementSender being called continuously by the change event. Calling

the function `CommandButton1_Click` has the same effect as clicking the **Send Message** button on the spreadsheet.

These examples within Excel should not be considered complete, although they work for the demonstration purposes of this redbook. If you intend to use them, you should add error checking to ensure that the data is handled correctly.

6.7 Summary

We have shown in this chapter a number of ways you could integrate Sametime functionality into your workflow applications. We have detailed how you can include announcement functionality into Notes and Domino applications. We then created an announcement Web service that can be accessed by any Web services client. We have shown this by using it from both a Notes agent and an Excel spreadsheet.

We hope that this chapter has given you some ideas on how to use Sametime from within your workflow applications. It is important to realize, however, that it should only be used when the situation merits it. You can imagine how annoying it would become to be constantly bombarded with instant messages and announcements from every application in the enterprise. We are not suggesting that Sametime should replace e-mail as the basis for your workflow applications, merely that Sametime could add a competitive advantage in the right scenario. Sametime does not replace e-mail; it complements it. Accordingly, you should find the balance that suits your application.



BuddyList service

This chapter discusses the BuddyList service component. This service is a wrapper for the storage service and is used only for handling the user buddylist. Using this service eliminates the need for a developer to know the exact format of the buddylist, as it saved on the server. Our purpose in this chapter is to present you with an easier way to perform the same actions of working with the storage service directly, but with less coding and with no internal knowledge regarding the attribute format in the server. The BuddyList service is available only for the C++ and Java APIs.

In this chapter, we will enhance the Extended Live Names sample, which is provided with the C++ Toolkit. We build upon the existing sample by describing how to perform the following additional actions using the BuddyList service:

- ▶ Loading the user buddylist after a successful login
- ▶ Handling the case where the buddylist does not exist on the server (for example, the first time that the user logs in)
- ▶ Saving the buddylist after every update by the user (for example, adding/removing people)

Note: For the purposes of this redbook, we will only work with people contained in private groups. (As will be explained later in the chapter, public groups are handled quite differently and therefore do not apply to this discussion.)

7.1 The BuddyList service advantages

As we describe in the introduction, the BuddyList service is a wrapper above the storage service, designed for more easily handling the loading and storage of the user buddylist. A developer working with the API still has the capability to manage the loading and storage of the buddylist using the storage service directly, so you may ask yourself why do I need this service?

The demand for the BuddyList service came from the requirement of developers wishing to use the same buddylist content that is used by the official Sametime clients (Desktop/Java connect) between various applications. Rather than managing multiple buddylists between applications, the API developers would like to work with the same buddylist content and be able to access this for their various applications. Accordingly, when users switch from one application to another, they can still get the same buddylist content. This capability has proven to be a very important requirement for Sametime customers.

While the storage service provides functionality to save or load attributes from the server, it does not recognize the attribute structure or values contained within the buddylist. It is the developer's full responsibility to know what the attributes contain.

7.1.1 The Sametime buddylist attribute structure

The official Sametime buddylist attribute ID is zero.

The official Sametime buddylist attribute value is a long string that contains the complete buddylist hierarchy and even the status of every group (open/close).

Example 7-1 is an example of the buddylist string value.

Example 7-1 Buddylist string value

```
Version=3.1.3/nG Work2 Work 0/nU CN=Yafit;Sami,0=itsoportall:: Yafit;Sami,/nU
CN=Bill;Mariscall:: Bill;Mariscal,/nU CN=Carl;Tyler,0=itsoportall::
Carl;Tyler,/nG Support2 Support 0/nU CN=camilo;lopez,0=itsoportall::
camilo;lopez,/nU CN=chris;heltzel,0=itsoportall:: chris;heltzel,/nG
CN=Administrators3 Administrators 0
```

As you can see from this example, the string is very complicated and the developer must know exactly how to parse it and add to this list in order not to damage the attribute.

This is exactly the problem that the BuddyList service is designed to solve. The BuddyList service saves the API developer the need to know this string structure and allows them to work with it in a more object oriented way.

One of the toolkit official samples is called Basic Buddy List. You can find the sample code as one of the samples in the C++ or Java APIs. We reference this sample, because it serves as a basis for comparison between using the storage service and using the BuddyList service. In the Basic Buddy List example, loading and saving the buddylist content from the server is accomplished by using the Storage Service directly. Our purpose in this chapter is to present you an easier way to perform the same actions with less coding and with no internal knowledge regarding the attribute format in the server by using the BuddyList service.

We encourage you to compare the code in this chapter to the code in the Basic Buddy List sample. This will be the best demonstration on how you can write the same code using two different services.

Our recommendation is to use the BuddyList service for handling the buddylist attribute and using the storage service for all other attributes.

The BL structure

The buddylist information is saved as a storage attribute on the server. The attribute ID is 0 and the value is a long list with spatial separators for every field. The BL structure represents the user buddylist. This structure saves you the need to know the string format. You will not have to parse the attribute string; instead, you will just need to use the documented object BL. This structure is a hierarchical representation of the user buddylist. The BL contains the buddylist groups. There are two kinds of groups:

- ▶ Private group: Represented by the class PrivateGroup.
- ▶ Public group: Represented by the class PublicGroup.

Both PublicGroup and PrivateGroup classes inherit the BLGroup class. The PrivateGroup class contains lists of people, and every person is represented by a BLUser class.

Note: A public group does not contain information about the list of people contained within the group. We do not save this information in storage, since a public group can contain hundreds of people and the content of the group may change while the user is offline. Accordingly, it is not meaningful to save this information. Instead, a client can always query a public group's content by using the LookupService.

7.2 Overview of the Extended Live Names sample

In this section, we review the functionality of the Extended Live Names sample. This sample is used as the basis for further customization, illustrating the functionality of the BuddyList service.

The objective of showing this sample is to illustrate how the user can add and remove people within the user list, but once the user logs out or quits the application, the buddylist content is not saved. Upon logging in again, the user will get an awareness list which contains only his or her name inside. This functionality will be enhanced by adding buddylist storage support.

The Extended Live Names sample is a stand-alone application that includes the following features:

- ▶ Log in to the Sametime community
- ▶ Add or remove users from the awareness list
- ▶ Initiate a chat with users who are logged in
- ▶ Log out from the Sametime community

7.2.1 Accessing the sample

To access the Toolkit pages that include the Toolkit documentation, samples, and binaries, follow these steps:

1. Navigate to the home page of your Sametime 3.x server
(http://<your_sametime_server_name>/stcenter.nsf)
2. Click the **SDK** link at the bottom of the page. You might need to scroll down the page to see the link.
3. Click **C++ Toolkit** on the Toolkits page. You are now at the Sametime 3.1 C++Toolkit home page, as shown in Figure 7-1 on page 167.

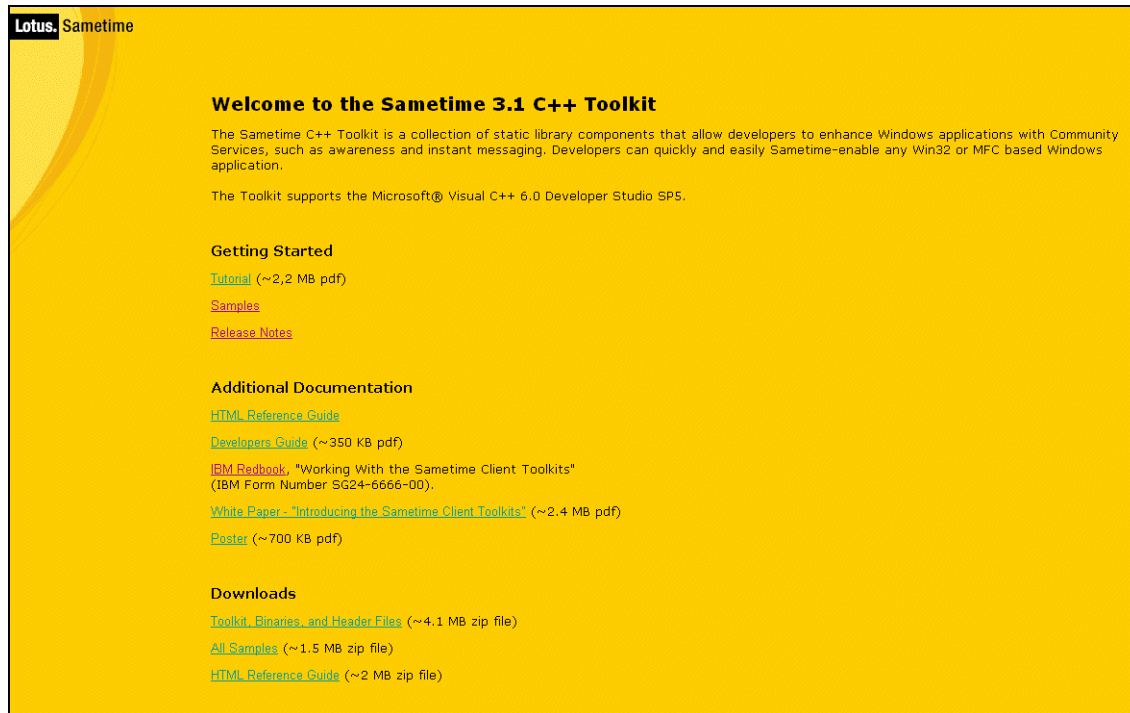


Figure 7-1 The Sametime 3.1 C++ Toolkit home page

7.2.2 Sample functionality overview

The following figures illustrate the functionality provided with the Extended Live Names sample in the C++ Toolkit.

Logging in to the Sametime community

Users can log into a Sametime community using the login dialog. This dialog includes three fields:

- ▶ The Sametime server name to which you wish to connect
- ▶ The user name
- ▶ The user password

Figure 7-2 on page 168 illustrates a user logging in.

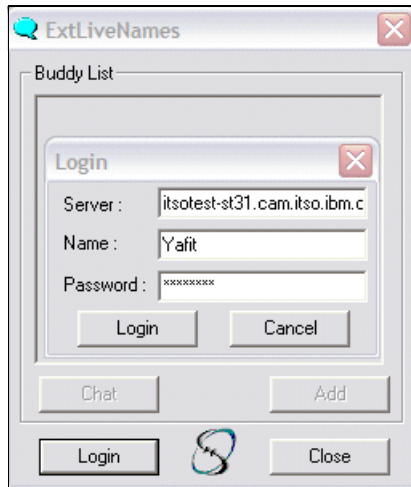


Figure 7-2 Login dialog

Adding people to my awareness list

Once user is logged in, the user can add people to his awareness list by clicking the **Add** button and entering the person's name. In the case of a name resolution conflict, this will display the Add Dialog box with a list of people matching the name criteria entered. The user can then select the desired name from the list, as shown in Figure 7-3.

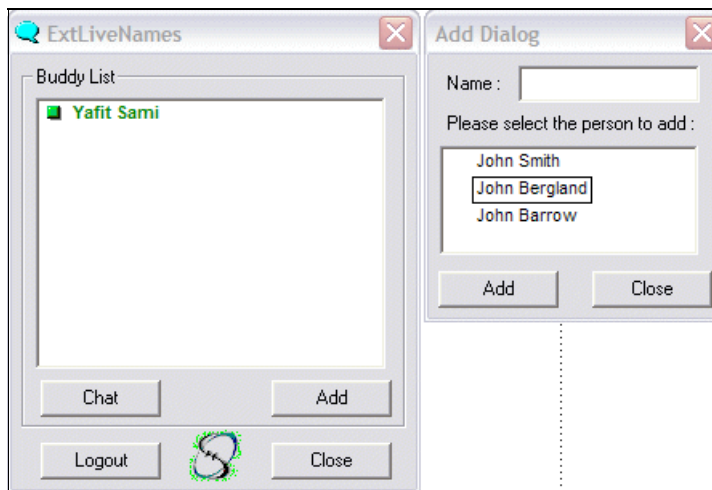


Figure 7-3 Add dialog displaying a list of names

Removing people from the awareness list

Once people have been added to the awareness list, the user can also remove people from the list by right clicking a name and choosing the **Remove** menu option.

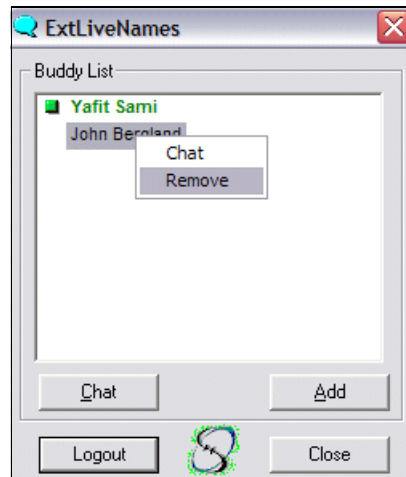


Figure 7-4 Remove feature

Instant messaging support

The user can start an instant chat with any online user in his list by performing any one of the following actions:

- ▶ Choosing a name and clicking on the **Chat** button
- ▶ Right-click on an online name and choosing the **Chat** menu option
- ▶ Double-click on an online name

Regardless of the method chosen, you should get a dialog box similar to Figure 7-5 on page 170 when you start the chat.

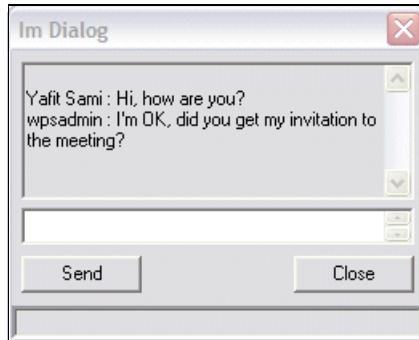


Figure 7-5 Chat dialog

Logging out from the Sametime community

Finally, the user can also log off from the community. The user only needs to click on the Logout button, as shown in Figure 7-6.

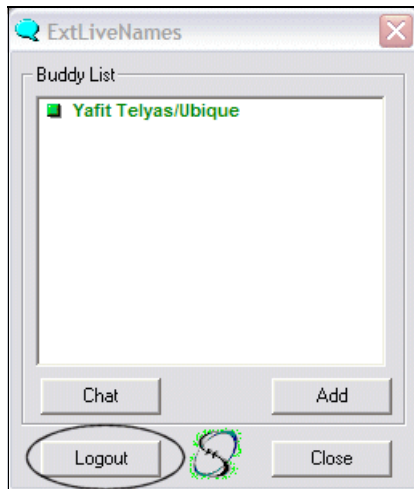


Figure 7-6 Logout

As illustrated, this sample code provides basic functionality for logging in, adding and removing people from the awareness list, and conducting basic chats. It is missing key functionalities, however, since the buddylist content is not saved when a user logs out or quits the application. Upon logging in again, the user will get an awareness list with only his name inside. In the next section, we will use the BuddyList service to modify this previous sample and make it much more useful.

7.3 Setting up the development environment

We are going to update the C++ Toolkit sample. This sample is developed using Microsoft's Visual Studio 6 IDE. As a prerequisite, there are some settings that need to be updated before adding the new functionality to the Extended Live Names project setting.

As a first step, the buddylist include path needs to be added to the sample:

1. Open the project setting dialog.
2. Choose the **C/C++** tab.
3. Choose the preprocessor category.
4. Add the buddylist include path to the “Additional include directory” field (see Figure 7-7).

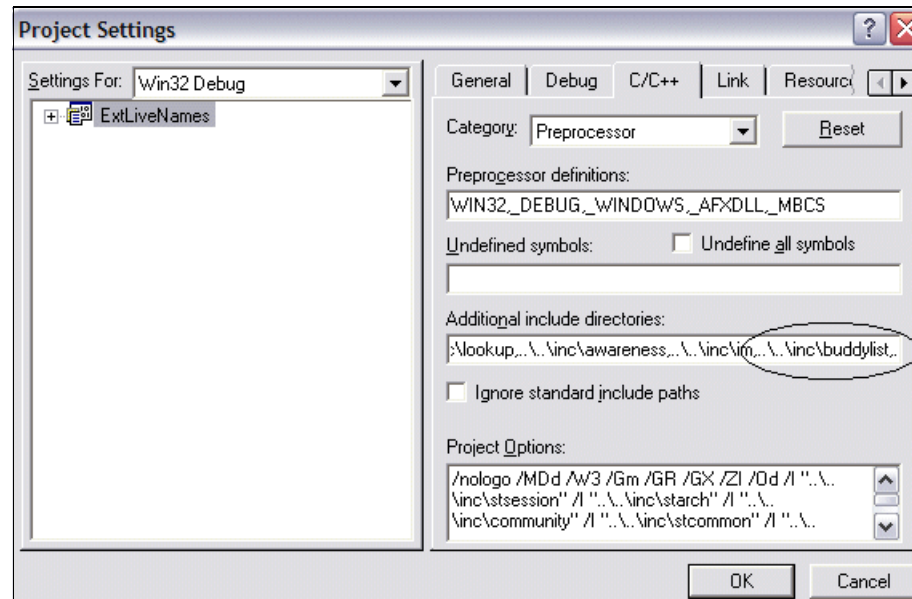


Figure 7-7 Adding buddylist include path

Next, we need to add the required libraries:

1. In the project setting dialog, choose the **Link** tab.
2. Choose the input category.
3. Add the libraries storage.lib and buddylist.lib to the “Object/library modules” field (Figure 7-8 on page 172).

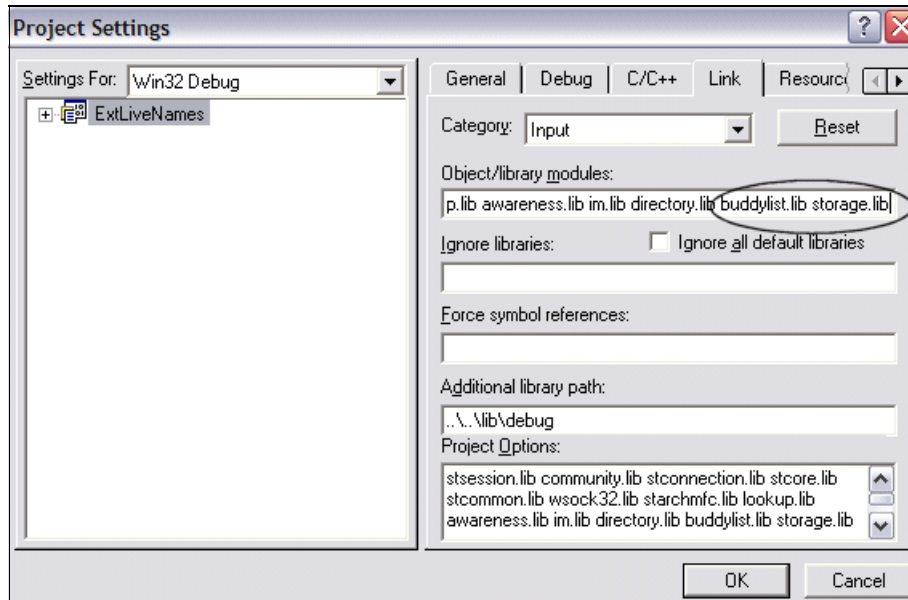


Figure 7-8 Adding library path

Once you have made these changes to Visual Studio 6, you are ready to begin modifications to the sample.

7.4 Loading the user buddylist after successful login

This section will describe how to add the load functionality to the sample using the BuddyList service.

7.4.1 Working with the BuddyList service

The ExtLiveNamesApp class contains a method called InitSametimeSession. This method is responsible for initiating all the services that we are using in the application. We will add the following line to this method to also initialize the BuddyList service:

```
new BLService(m_pSTSession);
```

The only class that will need to work with the BuddyList Service is the AwarenessList. This class is responsible for displaying the awareness list of the user and updating this list according to the user action (add/remove).

First, we will need to add a new member to this class that will represent the BuddyList service. To accomplish this, we add the following line to the class member definition:

```
BLSERVICE* m_pBuddyListService;
```

The AwarenessList class will also need to receive events from the BuddyList service so it will inherit the BLSERVICEListener class:

```
class AwarenessList : public CDialog,  
                     public BLSERVICEListener
```

For initializing the m_pBuddyListService member, we will add the lines in Example 7-2 at the end of the AwarenessList constructor.

Example 7-2 initial the m_pBuddyListService member

```
AwarenessList::AwarenessList(STSession* pSTSession, CRect windowRect,  
                             CWnd* pParent /* = NULL */) : CDialog(AwarenessList::IDD, pParent),  
    m_pAwarenessService(NULL),  
    m_pWatchList(NULL),  
    m_pAwarenessServiceListener(NULL),  
    m_pAwarenessStatusListener(NULL),  
    m_windowRect(windowRect),  
    m_bEnableRemoving(TRUE),  
    m_pChatUI(NULL)  
{  
    ...  
  
    if (pSTSession)  
        m_pBuddyListService = (BLSERVICE*)  
            pSTSession->GetComponent(BUDDY_LIST_COMP_NAME);  
    if (!m_pBuddyListService)  
        return;  
  
    m_pBuddyListService->addBLSERVICEListener(this);  
}
```

Finally, when the awareness list gets destroyed, it needs to stop listening to the BuddyList service event. To enable this, we will add the following lines in the OnDestroy method:

```
if (m_pBuddyListService)  
{  
    m_pBuddyListService->removeBLSERVICEListener(this);  
    m_pBuddyListService = NULL;  
}
```

7.4.2 Handling service available/unavailable events

The BuddyList service uses the Sametime server storage service. Server services may become unavailable while user is logged in for a variety of reasons. In our sample, we need to handle the case when the storage service is up or down.

Once the user logs in successfully, our sample should get one of the following events:

- ▶ BLSERVICEListener:serviceAvailable
- ▶ BLSERVICEListener::serviceUnavailable

If the user is logged in and the storage service goes down, the client will receive the serviceUnavailable event. When the storage service becomes available again, the client will receive the serviceAvailable event.

Let us examine how we handle each of these cases in the sample.

We declared a new boolean member in the class called m_BLSERVICEIsOn with a default value of FALSE. Now we will handle the service up/down events.

Upon receiving the service available event, we will reload the buddylist and change the m_BLSERVICEIsOn value to TRUE (see Example 7-3).

Example 7-3 Handling serviceAvailable event

```
void AwarenessList::serviceAvailable(BLEvent event)
{
    if (!m_BLSERVICEIsOn)
    {
        if (m_pBuddyListService)m_pBuddyListService->getBuddyList();
        m_BLSERVICEIsOn = TRUE;
    }
}
```

Note: Later in the chapter, we will describe how to handle the case when the user tries to add or remove people from their awareness list while the storage service is down. The user will be informed that any changes made to the user list while the service is down will not be saved and that the list will be reset with their original list on the server once the service comes up again. Section 7.5, “Handling a load failure” on page 177 provides additional details on this topic.

When the service is down, we set the `m_BLServicesOn` value to `FALSE` so that the application will know how to handle the case when the user tries to update his list (add/remove people) (see Example 7-4).

Example 7-4 Handling serviceUnavailable event

```
void AwarenessList::serviceUnavailable(BLEvent event)
{
    m_BLServicesOn = FALSE;
}
```

7.4.3 Loading the user buddylist

Now that we are all set, we can load the user buddylist after successfully logging in. The class `MainAppDlg` is responsible for receiving the logged in event. You can see in the original sample code that the application calls the `AwarenessList AddUser` method with the current logged in user information. We would like to change this and instead call a new method: `LoadBuddyList`. We will modify the original sample code by including this line:

```
m_pAwarenessList->LoadBuddyList(&myUserInstance);
```

instead of the original line:

```
m_pAwarenessList->AddUser(&myUserInstance);
```

We will add the new `LoadBuddyList` method to the `AwarenessList` class, as in Example 7-5.

Example 7-5 LoadBuddyList method

```
void AwarenessList::LoadBuddyList(STUser* user)
{
    m_pMyUserInstant = new STUser(*user);
    if ( (!m_BLServicesOn) && (m_pBuddyListService))
        m_pBuddyListService->getBuddyList();
}
```

As you can see in Example 7-5, there is a new class member called `m_pMyUserInstant`. This member contains the current logged in user information. We will need this member later in case of a load failure.

Note that we are loading the `BuddyList` service only if the service is up. If the service is not currently available, we will load it later on once the service becomes available. See the `HandlingServiceEvent` example (Example 7-3 on page 174) for reference.

If the service is available, we will call the `getBuddyList` method for loading the buddylist.

Assuming that the load is successful, we will receive the `BuddyList` service event `blRetrieveSucceeded` with all the information that we need. We will then over write this method in the `AwarenessList` class, as in Example 7-6.

Example 7-6 Retrieving the buddylist content

```
void AwarenessList::blRetrieveSucceeded(BLEvent event)
{
    if (m_pMyBuddyList) delete m_pMyBuddyList;
    m_pMyBuddyList = new BL(event.getBL());

    list<BLGroup*> myBuddyListGroups = m_pMyBuddyList->getblGroups();
    list<BLGroup*>::iterator itr;
    //Going over all the groups
    for (itr = myBuddyListGroups.begin(); itr != myBuddyListGroups.end();
itr++)
    {
        BLGroup* currGroup = *itr;
        //Adding people from privat groups
        if (typeid(*currGroup)==typeid(PrivateGroup))
        {
            PrivateGroup* pvtGroup = (PrivateGroup*)currGroup;

            list<BLUser*> users = pvtGroup->getUsersInGroup();
            list<BLUser*>::iterator userItr;
            for (userItr = users.begin(); userItr != users.end(); userItr++)
            {
                BLUser* blUser = (*userItr);
                STUser
currUser(STId(blUser->getBLId(),L""),blUser->getName(),blUser->getNickName());
                //Calling the AddUser with False value means that we just want to
                //add it to the UI but not re add t to our BL structure.
                AddUser(&currUser,FALSE);
            }
        }
    }
}
```

As you can see in the code from Example 7-6, we are using dynamic casting for determine which group the `BLGroup` object represents. In this example, we are handling only private groups. We will display all the people from our private groups in one flat list.

Next, we will save the BL structure in a class member called `m_pMyBuddyList`. We will use this member to save the buddylist in storage for every user change (adding or removing names).

Finally, after passing this method, the user will see all the people that have been added to his/her stored buddylist by this client or another Sametime client, such as the Sametime desktop connect client.

7.5 Handling a load failure

When trying to load the user buddylist content, we might get a fail response. This can happen if the user does not have a buddylist stored or if there is some kind of problem with the service. We would like to handle each of these two cases differently. In the case where the user does not have a buddylist, we will add his name to the awareness list as the original sample does. If there is some kind of problem with the service, we will display an error message so the user will be aware of the problem and contact his administrator for help.

We will handle the failure by overwriting the method `blRetrieveFailed`, as in Example 7-7.

Example 7-7 Handling a load failure

```
void AwarenessList::blRetrieveFailed(BLEvent event)
{
    if (event.getReason() == ST_ATTRS_NOT_EXIST)
    {
        AddUser(m_pMyUserInstant);
    }
    else
    {
        AfxMessageBox("There is a temporary problem with the storage service,
can't load your buddylist.");
    }
}
```

The error reason `ST_ATTRS_NOT_EXIST` indicates that this user does not have a buddylist stored. In this case, we will add the current logged in user (remember that we saved his information in the `m_pMyUserInstance` member) so the user will have an initial list.

Note: The `ST_ATTRS_NOT_EXIST` error is defined in the file `stcommon\inc\STError.h`

If the reason is different than ST_ATTRS_NOT_EXIST, we will display an error message to the user, such as the one shown in Figure 7-9.

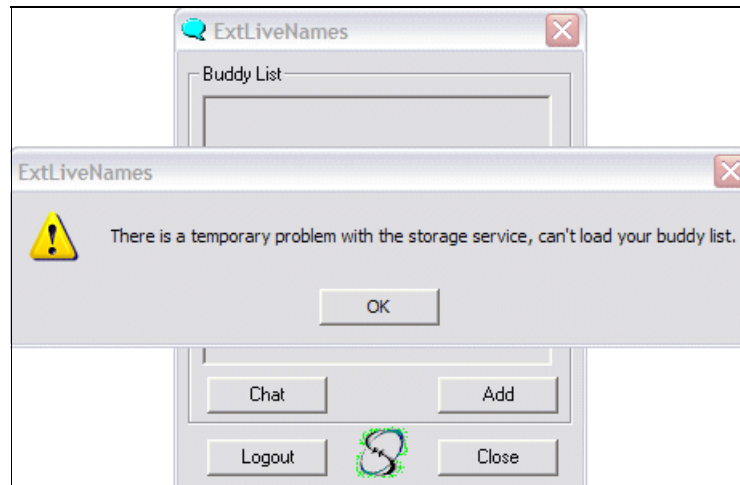


Figure 7-9 Displaying an error message when the server storage is down

7.6 Keeping the buddylist updated

Now that we have the buddylist content loaded, we will want to save it and keep it updated with any additional changes. As you may have noticed, the Extended Live Names application does not present groups. Instead, this application sample only allows you to add or remove people using a flat list. When adding people to the official buddylist, you must have them inside a private group. Remember that at the end of 7.2, “Overview of the Extended Live Names sample” on page 166 that the reason was provided for why it does not make sense to add people to a public group. For the purposes of illustrating functionality in this sample, we will therefore create a hardcoded group for our application called `ExLiveNamesGroup`. In the case of a private group, the group name and ID are identical. We will declare the group ID as a global variable at the top of the `AwarnessList.cpp` file by adding this line:

```
CString sampleGroupId = "ExLiveNamesGroup";
```

7.6.1 Adding a new user to a private group

When adding a new person, we would like to store the name inside the `ExLiveNamesGroup`. First, we will check whether or not this group exists. If not, we will create the group, add the new user, and then update the BL structure. This is demonstrated in Example 7-8 on page 179.

Example 7-8 Adding a new user

```
void AwarenessList::AddUser(STUser* user, BOOL newUser)
{
    if (!user || (FindItemByUserId(user->GetId()) != -1))
        return;

    STWatchedUser* pWatchedUser = new
    STWatchedUser(*user, STGroup(), STUserStatus(ST_USER_STATUS_DISCONNECTED, L""));
    if (!pWatchedUser)
        return;

    int pos = GetUsers();
    LV_ITEM item;
    item.mask = LVIF_PARAM;
    item.iItem = pos;
    item.iSubItem = 0;
    item.lParam = (DWORD)pWatchedUser;
    int index = m_ctlAwarenessListView.InsertItem(&item);
    m_ctlAwarenessListView.RedrawItems(index, index);
    m_ctlAwarenessListView.UpdateWindow();

    if (m_pWatchList)
        m_pWatchList->addItem(pWatchedUser);

    if (!m_BLSERVICE_IsOn)
    {
        AfxMessageBox("There is a temporary problem with the storage service,
        can't save your buddylist.");
        return;
    }

    if (m_pMyBuddyList)
    {
        list<BLGroup*> myBuddyListGroups = m_pMyBuddyList->getblGroups();
        list<BLGroup*>::iterator itr;
        BOOL groupFound = FALSE;
        STBLUser* newUser = new STBLUser(STId(user->GetId().getId(), L""),
            user->GetName(),
            user->GetDesc(),
            user->getNickName());
        for (itr = myBuddyListGroups.begin(); itr != myBuddyListGroups.end();
            itr++)
        {
            BLGroup* currGroup = *itr;
            if (typeid(*currGroup) == typeid(PrivateGroup))
            {
                PrivateGroup* pvtGroup = (PrivateGroup*)currGroup;
                CString currGroupId = pvtGroup->getBLId().c_str();
```

```

        if (currGroupId == sampleGroupId)
        {
            pvtGroup->addUser(newUser);
            groupFound = TRUE;
            break;
        }
    }
}
if (!groupFound)
{
    //there is no sample group, we need to add a new private group
    BSTR bstrId(sampleGroupId.AllocSysString());
    list<BLUser*> users;
    users.push_back(newUser);
    PrivateGroup* pvtGroup = new PrivateGroup(bstrId,bstrId,L"",true,users);
    m_pMyBuddyList->addGroup(pvtGroup);
    ::SysFreeString(bstrId);

}
m_pBuddyListService->setBuddyList(*m_pMyBuddyList);
}
}

```

If the user trying to add people while the BuddyList service is not available, the service will display a message indicating that this change will not be saved.

If the BuddyList service is available, we will search for the ExLiveNamesGroup. If we find it, we will add the new user to this group. Otherwise, we will create the group and add the user to the new group. The final step is to call the setBuddyList API method for storing the update list on the server.

If you log in from another application (for example, the Sametime Connect client), you will see a new private group called ExLiveNamesGroup with the user that you added.

7.6.2 Removing a person from the list

When removing a person from the list, we would like to immediately update the stored buddylist on the server. To accomplish this, we will update the AwarenessList::RemoveSelectedUser method.

First, we need to save the removed user ID before deleting the user object:

```

CString removedUserId(pWatchedUser->GetId().getId().c_str());
delete pWatchedUser;

```

We will use the user ID for searching for this user in our BL structure.

At the end of the `AwarenessList::RemoveSelectedUser`, we will add the lines shown in Example 7-9.

Example 7-9 Handling remove action

```
void AwarenessList::RemoveSelectedUser()
{
    ...

    if (!m_BLSERVICEIsOn)
    {
        AfxMessageBox("There is a temporary problem with the storage service,
        can't save your buddylist.");
        return;
    }

    if (m_pMyBuddyList)
    {
        list<BLGroup*> myBuddyListGroup = m_pMyBuddyList->getblGroups();
        list<BLGroup*>::iterator itr;
        for (itr = myBuddyListGroup.begin(); itr != myBuddyListGroup.end();
        itr++)
        {
            BLGroup* currGroup = *itr;
            if (typeid(*currGroup)==typeid(PrivateGroup))
            {
                PrivateGroup* pvtGroup = (PrivateGroup*)currGroup;
                list<BLUser*> users = pvtGroup->getUsersInGroup();
                list<BLUser*>::iterator userItr;
                for (userItr = users.begin(); userItr != users.end(); userItr++)
                {
                    BLUser* blUser = (*userItr);
                    CString currId(blUser->getBLId().c_str());
                    if (currId == removedUserId)
                        pvtGroup->removeUser(blUser);
                }
            }
        }
        m_pBuddyListService->setBuddyList(*m_pMyBuddyList);
    }
}
```

First, we need to check that the BuddyList service is available. If it is not available, the service will display an error message to the user indicating that his change cannot be saved at this time. If the BuddyList service is available, we will search for this user in all the private groups in the BL structure and remove it from the groups.

Note: The Extended Live Names sample is working with a flat list, so when the user deletes a person, we do not know which groups this person belong to. For this reason, we are deleting the user from all the private groups he is in. While this approach may be suitable for our sample, you would most likely wish to handle this differently in a real application. For example, you may want to give the user the list of groups that the person belongs to and then require the end user to choose which groups the person will be removed from.

After finishing the update to the BL structure, the only remaining step is to store the new buddylist content on the server. To accomplish this, we will call the API method `setBuddyList` with our updated BL structure. Once this is completed, the buddylist content is updated. You can confirm that the buddylist has been successfully updated by logging in from another client (for example, the Sametime connect client) and verifying that the user who was just deleted is no longer exists in the list.



Places and Place awareness

This chapter provides an in-depth study of the Sametime Places architecture. It reviews the concepts of Places, sections, activities, and attributes, then discusses the appropriate context and benefits for using them. To provide a meaningful context for discussing Places, it examines a detailed application that implements a multi-lingual panel discussion by leveraging Places, sections, Place awareness and features in the Sametime Community Server Toolkit.

Previous Redbooks have discussed the concepts and uses of Places and Place awareness. Chapter 4, “Places architecture”, in the redbook *Working with the Sametime Client Toolkits*, SG24-6666 discussed them from a client toolkit perspective. Chapter 3, “Places architecture”, in the redbook *Working with the Sametime Community Server Toolkit*, SG24-6667, discussed them from a server-side perspective. This goal of this chapter is to expand upon these existing discussions of Sametime Places architecture, and to provide an new, meaningful example which illustrates how to leverage this Sametime technology.

8.1 Key concepts within Places architecture

Before we begin examining the scenario and application in this chapter, it is important to provide a basic understanding of the key concepts used within the Sametime Places architecture. Once these concepts are introduced, subsequent sections illustrate how they fit within the specific scenario and why each component is a very important piece of the overall solution.

8.1.1 What are Places and why use them?

A Place is a virtual space within Sametime that allows users who “enter” it to:

- ▶ Be aware of the online status of other users in the Place
- ▶ Communicate with other members in a number of ways
- ▶ Take part in different activities that run within the Place

Technically speaking, a Place is defined by a name and a Place type. The name of a Place is just a string and the Place type is an integer. The application developer chooses the appropriate name when the Place is created.

8.1.2 What are sections and why use them?

In the Sametime Places model, a Place is made out of sections. This makes a Place similar to a natural Place that has different rooms inside it. Every user that enters a Place enters into a specific section in that Place. This is very useful; for example, in our upcoming scenario application, we separate the audience members from the panel by putting them into different sections. The concept of sections further enhances the awareness and communication capabilities inside a Place by adding the notion of scope.

8.1.3 What are activities and why use them?

Activities are server-side applications that participate in a Place just like users and sections. Activities can communicate with users and users can communicate with activities. However, activities have more powerful permissions. They are a super-user within the Place. An activity can communicate with any section and any user within a Place and change the attributes of any user or section within the Place. Within the context of the upcoming panel discussion example, the activity is responsible for several important tasks.

8.1.4 What are attributes and why use them?

All Place members (that is, users, sections, and activities) can have attributes. Attributes are key/value pairs that can contain any data/information that the developer wishes to attach to the given member. Other members can listen for attribute changes and react accordingly.

8.2 Scenario

The application is designed to mimic a multi-national, multi-language panel discussion. The panel discussion consists of a panel of experts, audience participants and a specially designed auditorium. Each panel expert enters through the lobby and is then escorted to the stage of the auditorium. Audience participants also enter the lobby and are then escorted into a specific country section based on a Java properties file. In the application, we have defined four country specific sections: America, Germany, France, and UK.

Any audience member can ask a question. When they do, they enter a Queue with other participants and then wait for their turn. Audience members are able to ask questions in the order in which they entered the Queue. When their turn arrives, they ask the panel of experts their question. The question is translated and then sent to the panel and then to each section in each section's native language. Any panel expert is capable of responding to the question. Their response is also translated and then sent to each section, again in their native language. (In our example, we made an assumption that all panel members speak English.) When the audience member is finished asking their question, they return to their section in the auditorium and the next person in the Queue can ask their question.

8.3 Application overview

Figure 8-1 on page 186 provides an overview of how the application maps to the Sametime Places model.

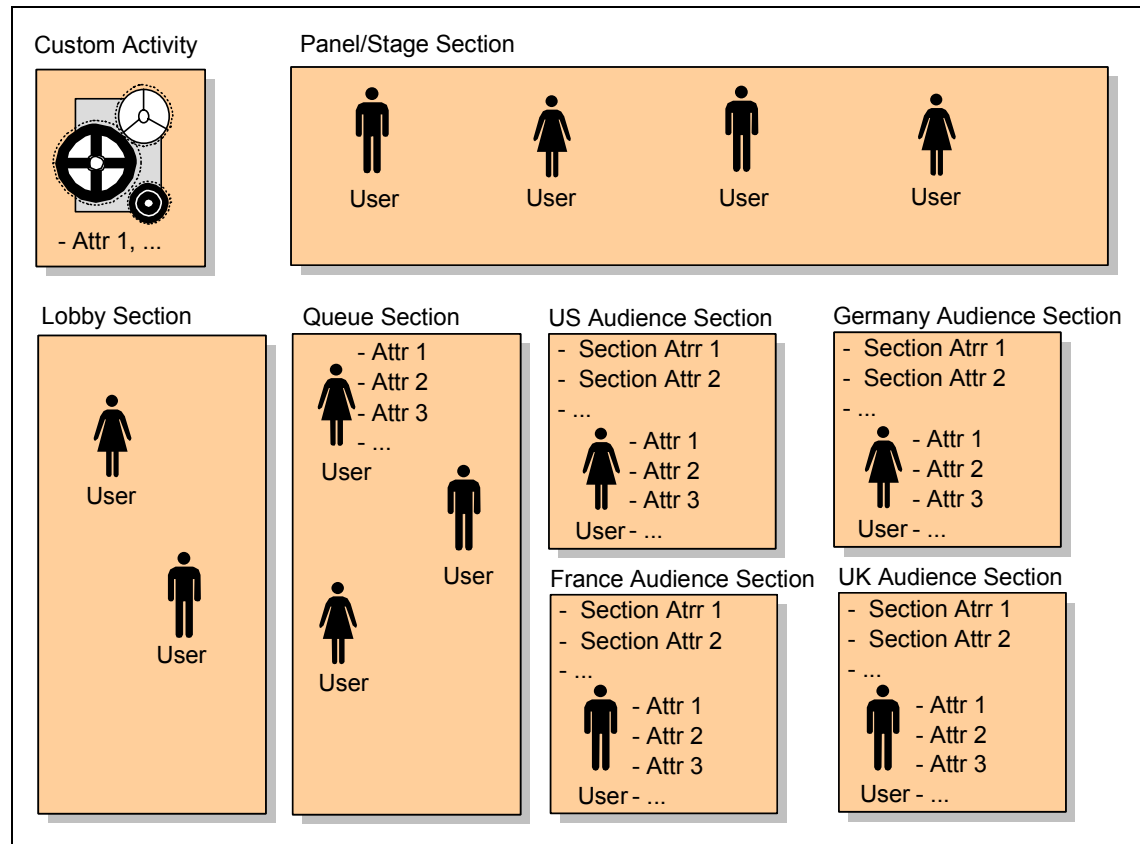


Figure 8-1 High-level overview of panel discussion

The Place in our application consists of seven sections, one activity, and a variable number of users. The Lobby section is the section where all users (panel or audience) enter. The application monitors the Lobby, which is where all users enter, and moves them into either the Stage section or one of the Audience sections. The Stage section is a special Sametime section where all the panel members reside. There are four audience sections, one for each country that we predefined in the `RedPanelDiscussionConstants` Java class. The four countries we predefined were America, France, the United Kingdom, and Germany. (These sections were chosen because the translation engine was capable of translating between these three languages.) The final section is the Queue section, which is where audience members line up to ask questions. The custom activity is a Sametime activity that translates messages into different languages and sends the translated messages to the appropriate sections. In addition, it monitors the Lobby for new users entering the Place, and monitors the Queue for the next person to ask a question.

8.3.1 Applying Places to the context of the scenario

As mentioned above, a Place is a virtual space within Sametime. It allows for users who “enter” it to be aware of others’ online status, communicate directly with other members of the Place, and take part in activities that run within the Place.

In the Panel Discussion, we use a Place to provide an area for the activity and application to work within. By using the Place, we also have the ability to add listeners to allow us to listen for users entering and leaving the Place.

8.3.2 Applying sections to the context of the scenario

A Place is made out of sections and each user that enters a Place enters into a specific section in that Place. This is very useful; for example, in our application we separate the audience members from the panel by putting them into different sections. This allows the audience to communicate among themselves without having the presenters “hear” them. This is possible due to the fact that sections are treated in the Places model just like any other Place member. A special section in the Place is entitled the Stage. A user in the stage has more permissions than others. For example, he can talk to the entire Place, not only to other users in his section.

Note: The Stage section is a special section with special abilities. For a full list of permissions in a Place see Table 3-1 on page 55 in the redbook *Working with the Sametime Client Toolkit*, SG24-6666.

Our example divides the application into six non-stage sections and one stage section. Each section has different capabilities and attributes within the Place. Table 8-1 on page 188 describes the different sections, their capabilities and their attributes.

Table 8-1 Sections in the application

Section name	Description
Stage	<p>Only panel members are placed into this section. The panel members are defined in the RedPanelDiscussion.properties file.</p> <p>The example does not use Sametime's ability to restrict who can enter a section. Instead, entrance to the section is controlled by a custom activity that will be discussed later in this chapter.</p> <p>The main use of this section is to separate the panel members from the audience. We do not use any of the special communication capabilities of the stage section.</p>
Lobby	<p>All users entering the Place enter this section by default. Technically, the Lobby is defined as the lowest numbered non-stage section in the Place. By default, Sametime puts all users into this section. This feature allows the application to monitor who enters the section and then moves them into the appropriate section or, alternatively, just leaves them in the lobby.</p> <p>Within the context of the application, users in the Lobby have no interaction with the Place. The only capabilities they have are to wait to be moved into a section.</p> <p>The main use of this section is to control who enters the Place. This section could also be used to assigned special attributes to users when they enter the Place.</p>
Queue	<p>Users who want to ask a question are placed in this section. When in the Queue section, they are aware of other members in the Queue section. They, however, are not able to communicate with members from their country section because they have physically moved into a different section within the Place.</p> <p>The Queue section is not a special section in Sametime. The application simply makes the first non-lobby, non-stage section the Queue section.</p> <p>The main use of this section is provide the application with an easy way to manage and monitor users who are asking questions.</p>

Section name	Description
Audience Section	<p>Users are placed into these sections based on a Java properties file. (This is just an example. Another possibility is to look up an attribute in a directory or some other persistent store.)</p> <p>Members in a particular audience section can participate in a chat area with other members in the section. They can also elect to ask a question. When a user elects to ask a question, they are moved from this section and put into the Queue section.</p>

8.3.3 Applying activities to the context of the scenario

Activities are server-side applications that can act as super-users within the Place. They can communicate with any section and any user within a Place and change the attributes of any user or section within the Place.

In the panel discussion example, the activity is responsible for several tasks:

- ▶ Monitoring users entering the lobby and moving them into sections
- ▶ Managing the ask a question Queue
- ▶ Translating and distributing messages

An activity was used because users in sections needed the ability to send a message to users on the stage and users in other sections after being translated into several languages.

8.3.4 Applying attributes to the context of the scenario

Attributes are key/value pairs that can contain any data/information about a Place member. Other members can listen for attribute changes and react accordingly.

In the Panel Discussion, we use attributes to specify certain attributes, such as the language preference and name for sections and users. The example does not take advantage of Sametime's ability to listen to attribute changes.

8.4 Setting up and running the application

The application consists of a Sametime server application and client-side applet. This section describes how to set up and run the application.

Attention: All files referenced for installing the application are available for download from the Redbooks Web site. See Appendix C, “Additional material” on page 467 for specific instructions on downloading the sample code.

8.4.1 Assign users to sections and the panel

In order for the application to work properly, you need to modify a properties file with the names of the people who belong in each section and the stage.

As mentioned earlier, the application contains four country specific sections and a panel section. The RedPanelDiscussion.properties file contains a name/value pair for the names of the members who should be placed in the country specific sections. There is also a name/value pair for who the panel members should be.

```
America.names=Scott Rice,Bill Tworek,John Bergland  
England.names=Kate Covey,Chameen Letourneau,E.J. Ward  
Germany.names=Mike Uson,Mike O'Donnell,John Majerowski  
France.names=Chuck Gullett,Paul Letourneau  
Panel.names=Jonas Covey,John Barrow,Washington Cabral,Carl Tyler
```

8.4.2 Download and detach files

You must download and detach the server application and client/Applet.

Server application

1. Download RedPanelDiscussionSA.ZIP.
2. Extract the files into a directory on the computer where you want the server application to be hosted.

Client/Applet

1. Download RedSamples.ZIP.
2. Extract all files, using the Use Folder Names option, to the HTML directory of your Sametime server.

8.4.3 Environment variables

The server application needs the following JAR files to be included in the CLASSPATH environment variable:

- ▶ stcommsrvrtk.jar
- ▶ activation.jar (Used by Web Service)
- ▶ commres.jar (Used by Web Service)
- ▶ mail.jar (Used by Web Service)

- ▶ soap.jar (Used by Web Service)
- ▶ xerces.jar (Used by Web Service)

The applet needs the following JAR files to be included in the CLASSPATH environment variable:

- ▶ STComm.jar

The server application also requires that the directory containing java.exe be included on the PATH environment variable.

Attention: Each of these jar files referenced above are available for download from the Redbooks Web site. See Appendix C, “Additional material” on page 467 for specific instructions on downloading the sample code.

8.4.4 Sametime server preparation

When developing Sametime applications, it is often easiest if you install a Sametime server on your development machine (though you should not test your application with such a setup). However, if your Sametime server is on another machine than your development machine, you must tell the Sametime server that it is OK to accept IP connections from server applications on different machines.

In previous versions of Sametime, you would tell Sametime to accept IP connections by modifying the Sametime.ini and adding a VPS_Trusted_IPS entry within the ini config section. However, in Sametime 3.0, this method will not work. Instead, the trusted IP address must be set in the stconfig.nsf database. The change is made on the Community Connectivity tab by entering the trusted IP addresses, separated by either commas (,) or semicolons (;) in the CommunityTrustedIP field.

8.4.5 Start the server application

To start the server application, open a command window and switch to the directory where you extracted the ZIP file. Type:

```
StartPanelDiscussionSA fully qualified host name
```

This will start a batch file that will execute the server application.

To shut down the server application, click the **Shut Down Application** button in the applet window that pops up after the server application starts.

8.4.6 Start the client applications

To start the client applet:

1. Open a Web browser and go to `http://<sametime_server>/redsamples/`.
2. Click on the **Panel Discussion Applet** link.
3. When prompted, enter your user name and password and then click on the **Log On** button.

8.4.7 Understanding the client-side

The client-side of the application consists of Java applet based on the Places applet from the Sametime Java Client Toolkit samples. The applet provides the user interface for both the audience and panel members. This section discusses the applet and what it does.

Note: Since the applet is based on an sample included with the Sametime Java Client Toolkit, this section will only touch upon enhancements or changes to the code that directly relate to the Panel Discussion application and applet.

8.4.8 Using the Applet GUI

Figure 8-2 on page 193 show the Panel Discussion applet from an audience member's perspective.

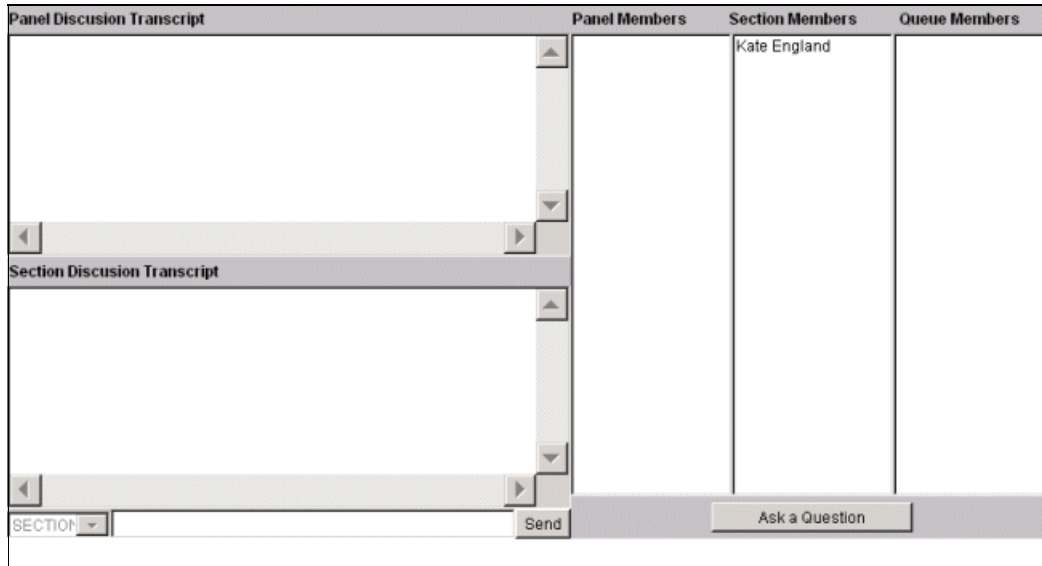


Figure 8-2 Panel Discussion applet: audience member perspective

The Panel Discussion Transcript area is where all questions and answers will be displayed, translated into the user's native language (as defined by the section they are a member of).

The Section Discussion Transcript area is where the section/country specific discussion is displayed. Each country section has its own distinct transcript area that only members of that section/country can see and participate in.

The Panel Members area lists the members in the Panel (stage) section. This area simply shows who is in the Panel section and is not enabled for instant messaging.

The Section Members area lists the members who are in the users current section. The list is not enabled for instant messaging.

Note: If the user is in the Queue, then this section and the Queue members sections will be exactly the same.

The Queue Members area lists users in the Queue section who are waiting to ask a question. The users are listed in the order in which they entered the Queue. The list is not enabled for instant messaging. Also, when a user enters the Queue, the Section Discussion Transcript area is disabled.

When a user wants to send a message to their country section, they type their text into the input text box to the left of the Send button and then click the **Send** button. The text message will be sent to all members in the section without any translation. (The applet assumes that the user sending the message and the other members from their country speak the same language.)

When a user wants to ask a question, they click on the **Ask a question** button located near the bottom-right corner of the applet. If they successfully enter the Queue, the button text will change to “Return to Section”. Or if they are the first member in the Queue or it is their turn to ask a question, it will say “No more questions”. When a user is in Queue, they cannot participate in their country section discussion or the Queue discussion. Basically, the user waits for their turn to ask a question. If the button text says “No more questions”, the user can send as many questions as they would like to the panel by typing their question in the input box and clicking the **Send** button. When they are finished, they just click the **No more questions** button. The user will be returned to their country section.

Figure 8-3 shows the Panel Discussion applet from a panel member’s perspective.

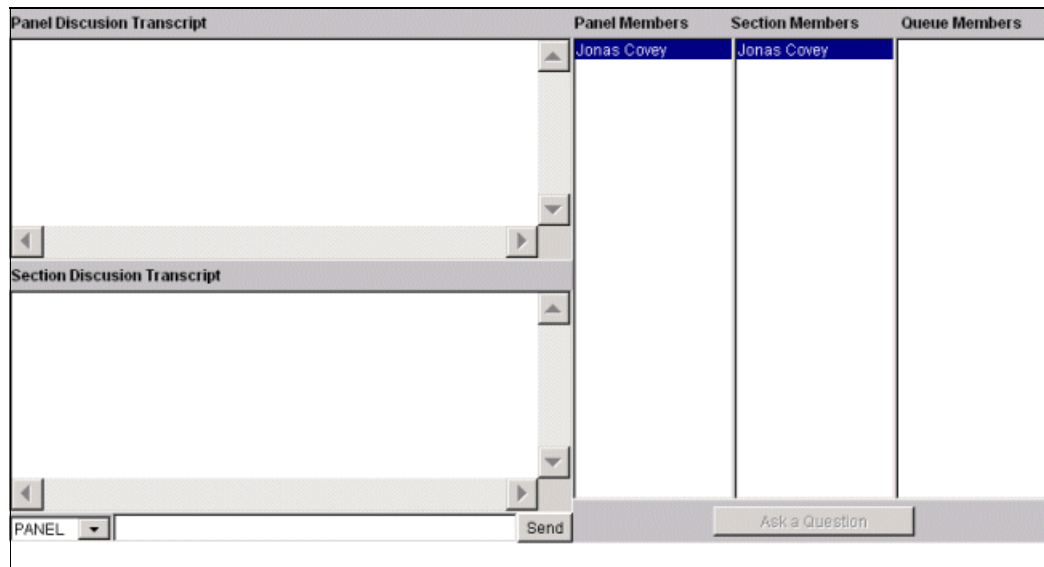


Figure 8-3 Panel Discussion Applet: panel member perspective

The applet from the panel member perspective has the same user interface widgets as the audience member, except that they are wired a little differently. One difference is that the drop-box in the bottom left-hand side of Figure 8-3 is

enabled. This drop-box allows the panel member to send a message to either the panel section discussion or panel discussion (the Place) via the activity. Text messages sent to the panel section discussion are not translated, but those sent to the panel discussion (Place) are. The other difference is that panel members do not have the ability to ask questions.

8.4.9 Entering the Panel Discussion Place

Once the user has logged into the Sametime server and the `loggedIn()` event has been fired, we are ready to enter into the Panel Discussion Place that was created by our server application. A Place is uniquely identified by a combination of two attributes: the Place name and the Place type. If you recall in 8.5.2, “Creating a persistent Place and Place characteristics” on page 206, we used the Server Application Service to create a persistent Place with a Place name of “RedDiscussionPanel” and a Place type with a value of 10010. The following excerpt shows how we use these same two values to enter the Place using the Java Client Toolkit:

```
public void loggedIn(LoginEvent event)
{
    // Get reference to Place we want to enter
    m_place = m_placesService.createPlace(PLACE_NAME,
                                         "Red Panel Discussion",
                                         EncLevel.ENC_LEVEL_DONT_CARE,
                                         PLACE_TYPE);

    // Add an adapter to be the listener to this Place.
    addPlaceListener();
    // enter place in a section
    m_place.enter("", PlacesConstants.PLACE_CREATION_JOIN , false);

    m_mySTUserInstance = event.getLogin().getMyUserInstance();
}
```

Note that in the `enter()` method we used the `PlacesConstants.PLACE_CREATION_JOIN` constant to indicate that we want to join an existing Place rather than create one. We also requested that we enter into a non-stage section. If the user successfully enters the Place, the `entered()` event will be fired.

Tip: All users who are in the non-stage section will all enter into the same section. The section will always have the lowest Member ID of all the non-stage sections.

8.4.10 Getting a reference to the activity

Since we set the default activity for our Place using the server application service, the `activityAdded()` event will be fired shortly after a user enters the Place. The following code checks to see if the activity is of the same type as the one we created in our server application. If it is, we store a reference to activity so we can send messages to it later.

```
public void activityAdded(PlaceEvent event) {
    Activity activity = e.getActivity();

    if (event.getActivityType() == ACTIVITY_TYPE) {
        m_specialActivity = event.getActivity();
    }
}
```

8.4.11 Getting references to sections

At one point or another, the applet is responsible for moving the users into either the Queue or Lobby sections. Therefore, we need to store a handle for each of these sections. The only time the applet can get a reference to these sections is when the `sectionAdded()` event is triggered, which occurs shortly after a user enters the Place. Since these sections are just regular sections that only have meaning within the application, we need a way to determine which section is which. There are a couple of approaches we could have employed. For one, the server application could have set an attribute on each section. Then when the applet received the event, it could have checked the attribute to discover which section was which. A second approach would have been to send the activity a request for the Member ID numbers of each of these sections using the `sendData()` method. The activity could have then replied using the `sendData()` method and the applet could have responded accordingly. While both these approaches have their merits and deficiencies, we choose to implement a different approach using an algorithm similar to the one the server application uses. Example 8-1 shows the algorithm we used.

Example 8-1 sectionAdded() event in PanelDiscussionApplet.java

```
public void sectionAdded(PlaceEvent event)
{
    Section tempSection;
    Section newSection = event.getSection();
    Integer sectionId = newSection.getMemberId();
    String sectionKey = "Section" + sectionId.toString();

    m_sections.put(sectionKey, newSection);
    m_chSections.add(sectionKey);
}
```

```

        if (newSection.isStage()) {
            newSection.addSectionListener(new PlaceSectionAdapter(m_panelList));
        } else if (m_lobby == null) {
            m_lobby = newSection;
        } else {
            if (newSection.getMemberId().intValue() <
m_lobby.getMemberId().intValue()) {
                tempSection = newSection;
                newSection = m_lobby;
                m_lobby = tempSection;
            }

            if (m_queue == null) {
                m_queue = newSection;
                newSection.addSectionListener(new
PlaceSectionAdapter(m_queueList));
            }
        }
    }
}

```

The event, which is triggered each time a section is added to the Place, first checks to see if the section is the special Sametime stage section. If so, we add a section listener on the stage so that the stage member awareness list in the applet is kept current. If not, we check the `m_lobby` member variable to see if it has been set yet or not. If it has not, we assume that this section is the Lobby and set the `m_lobby` member variable equal to the new section. The lowest numbered section, as defined by Member ID, is the section all members enter into a Place by default. We are calling this section the Lobby. If `m_lobby` has already been set, we check to see whether or not the Member ID for the section that is being added is lower than the Member ID of the section defined by `m_lobby`. If so, we assign the new section to the Lobby and re-process the section that was defined by `m_lobby`. This check is performed because it is possible that the sections will arrive out of order. If the Member ID of the section being added is greater than the Member ID of the Lobby, then we check to see if the `m_queue` member variable, or the Queue section, has been set. If not, we decided this section would be considered the Queue and therefore set the member variable `m_queue` equal to the new section being added. Finally, if none of these conditions are met, we do nothing. We could potentially track all sections in the Place, but we decided that it was not necessary for this example.

8.4.12 Entering the Queue

When a audience member wants to ask a question, they first need to enter the Queue. To enter the Queue, they click the **Ask a question** button, which triggers the `actionPerformed()` event. The code for this event, listed in Example 8-2 on page 198, moves the user into the Queue section of the Place.

```
public void actionPerformed(ActionEvent event)
{
    PlaceMember receiver = null;

    // check which object triggered this event
    if (event.getSource() == m_btnSend)
    {
        // code removed
    } else if (event.getSource() == m_btnQueueInOut) {
        // button under list box (Ask a question) button triggered the event
        if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_INSECTION)) {
            // ask a question by entering the queue
            m_myself.changeSection(m_queue);
        } else if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_INQUEUE)) {
            // did not ask a question, but want to leave
            m_myself.changeSection(m_lobby);
        } else if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_FINISHED)) {
            m_myself.changeSection(m_lobby);
            m_chScope.select(0);
        }
    }
}
```

When the user moves into the Queue section, they are put into a Queue in the server application, which is implemented using the `java.util.Vector` class. If no users are in the Queue when the user enters the section, the server application sends a data message to the user notifying them that they can start asking their question or questions. Otherwise, the user waits in the Queue in the order in which they entered. When it is their turn, they will be notified with a data message.

Note: In the server application, we choose to use a data message over a text message or user attribute change for a few reasons. For one, we wanted to show an example of using the data message capabilities of the APIs. Secondly, as a best practice, we wanted to differentiate program control messages from text messages.

When server application sends the data message to the user letting them know it is their turn to ask questions to the panel, the `dataReceived()` event will be fired in the applet. The code then checks to make sure that data type of the event is of type `DATA_TYPE_SPEAK` and then rewires the applet to send text messages to the activity instead of the section. Example 8-3 on page 199 shows the code in the `dataReceived()` event.


```
public void dataReceived(MyselfEvent event)
{
    NdrInputStream in = new NdrInputStream(event.getData());
    int data_type = event.getDataType();

    switch (data_type) {

        case DATA_TYPE_SPEAK:
            m_chScope.select(1);
            m_btnSend.setEnabled(true);
            m_btnQueueInOut.setLabel(BTN_TEXT_FINISHED);
            break;
        case DATA_TYPE_ISPANEL:
            // code removed for example
            break;
        default:
            break;
    }
}
```

When this event is fired, a `MyselfEvent` is passed in as an input parameter. From this object, we extract the data type of the message. A data type is simply an integer that represents a type of message. The data type integer by itself has no meaning until some code interprets the data type and takes some action based on it.

8.4.13 Sending a message to the panel

To send a question to the panel, the user types their question in the text input box located at the bottom of the applet to the left of the Send button and then clicks the **Send** button. When they click the **Send** button, the `actionPerformed()` event is triggered. In this event, we send a text message to the activity so it can be translated and then distributed to each section in their native language and to the panel in English. Sending a text message to an activity is exactly the same as sending to an other `Place` member. You simply get a handle to the `Place` member and use the `sendText()` method. Example 8-4 shows how to send a text message to an activity.

```
public void actionPerformed(ActionEvent event)
{
    PlaceMember receiver = null;
    String fromLanguage = "";
    STAttribute myAttrib = null;
```

```

Enumeration myAttribs = null;
String text = "";
String scope = "";

// check which object triggered this event
if (event.getSource() == m_btnSend)
{
    // send button triggered event
    text = m_tfSend.getText();
    scope = m_chScope.getSelectedItem();

// determine whether the SECTION or ACTIVITY/PANEL should receive the message
    if (scope.equals("SECTION"))
    {
        receiver = m_mySection;
    }
    else if (scope.equals("PANEL")) {
        receiver = m_specialActivity;
    }
    else
    { }

    if (receiver instanceof Activity) {
        myAttribs = m_myself.getAttributes();
        myAttrib = null;
        fromLanguage = "";

        while (myAttribs.hasMoreElements()) {
            myAttrib = (STAttribute) myAttribs.nextElement();

            if (myAttrib.getKey() == SECTION_LANGUAGES_KEY) {
                fromLanguage = myAttrib.getString();
                break;
            } // end if
        } // end while

        fromLanguage = (fromLanguage.equals("")) ? DEF_LANG : fromLanguage;

        m_myself.changeAttribute(new
STExtendedAttribute(MY_DISPLAY_NAME_KEY, m_myself.getDisplayName()));

        receiver.sendText(text);

    } else {
        receiver.sendText(text);
    }

    m_tfSend.setText("");

```

```

    } else if (event.getSource() == m_btnQueueInOut) {
// button under list box (Ask a question) button triggered the event
    if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_INSECTION)) {
        // ask a question by entering the queue
        m_myself.changeSection(m_queue);
    } else if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_INQUEUE)) {
        // did not ask a question, but want to leave
        m_myself.changeSection(m_lobby);
    } else if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_FINISHED)) {
        m_myself.changeSection(m_lobby);
        m_chScope.select(0);
    }
}
}
}

```

After the text message is sent, the applet listens for the translated message to be sent back by the activity. Once received, it will add the message to the Panel Discussion Transcript text area.

8.4.14 Leaving the Queue

In the Panel Discussion example, a user can be in two states when they decide to leave the Queue. They could either be finished asking their questions or they could have decided that they did not want to ask their question anymore. Either way, when the user chooses to leave the Queue, they click on a button. The button will either be labeled “No more questions” or “Return to Section”. When clicked, the button will trigger the `actionPerformed()` event. Instead of trying to remember the section the user came from, we decided to take a little short cut and just sent them back to the Lobby. When the user re-enters the Lobby, the server application will move them back into the appropriate section. Example 8-5 shows the relevant process.

Example 8-5 Leaving the Queue

```

public void actionPerformed(ActionEvent event)
{
    PlaceMember receiver = null;
    // check which object triggered this event
    if (event.getSource() == m_btnSend)
    {
        //
        // code removed from example
        //
    } else if (event.getSource() == m_btnQueueInOut) {
// button under list box (Ask a question) button triggered the event
        if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_INSECTION)) {
            // ask a question by entering the queue

```

```

        m_myself.changeSection(m_queue);
    } else if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_INQUEUE)) {
        // did not ask a question, but want to leave
        m_myself.changeSection(m_lobby);
    } else if (m_btnQueueInOut.getLabel().equals(BTN_TEXT_FINISHED)) {
        m_myself.changeSection(m_lobby);
        m_chScope.select(0);
    }
}
}
}
}

```

8.4.15 Logging in as a panel member

Since audience and panel members use the same applet to access the application, the applet needs a way to determine which perspective to display to the user. The applet determines this by listening for a data message of the `DATA_TYPE_ISPANEL` type. When it receives the data message, it examines the content in the data byte array that was passed with the event object. If the value of the data is equal to the integer 1, then a member variable called `isPanelMember` is set to true; otherwise, it is set to false. In addition to showing how this is accomplished, Example 8-6 also shows how to read data using an undocumented Sametime class called `NdrInputStream` in the `com.lotus.sametime.core.util` package.

Note: Panel member names are listed in a properties used by the server application.

Example 8-6 Determining if a user is a panel member

```

public void dataReceived(MyselfEvent event)
{
    NdrInputStream in = new NdrInputStream(event.getData());

    int data_type = event.getDataType();

    switch (data_type) {

    case DATA_TYPE_SPEAK:
        // code removed for example
    case DATA_TYPE_ISPANEL:
        int i = 0;
        try {
            i = in.readInt();
        } catch (IOException e) {
        }
    }
}

```

```

        this.isPanelMember = (i == 1) ? true : false;

        if (this.isPanelMember == true) {
            m_btnQueueInOut.setEnabled(false);
            m_chScope.select(1);
            m_chScope.setVisible(true);
            m_chScope.setEnabled(true);
            m_chScope.repaint();
        }
        break;
    default:
        break;
    }
}

```

Note: In Example 8-3, we used an undocumented class named **NdrInputStream**. The class extends the `java.io.DataInputStream` class which is part of the standard JDK. It provides a easier way to read primitive data types from the input byte array in a machine-independent way.

8.5 Building the server-side

The server-side of the application is implemented as a Java application that acts as a Sametime server application and activity. The application is responsible for:

- ▶ Creating a Panel Discussion Place as a persistent Place
- ▶ Registering itself as an activity
- ▶ Responding to requests for and sent to the activity
- ▶ Managing the question Queue

This section walks through how the server-side functions

8.5.1 Logging in as a server application

The first step to logging in as a server application is to establish a Sametime session. But before we start the session, we load the Sametime components we will be using in the application. The `ServerAppService` component is loaded to allow us to log into the Sametime server as a Server Application. The `ActivityService` component is loaded to allow us to create an Activity. And finally, the `PlacesAdminService` components are loaded to allow us to create a persistent Place and set Place characteristics. Example 8-7 on page 204 shows the code.

Example 8-7 Establishing a Sametime session

```
public RedPanelDiscussionSA(String host) {
    try
    {
        m_session = new STSession("Red Panel Discussion SA" + this);

        String [] compNames = { ServerAppService.COMP_NAME,
                                ActivityService.COMP_NAME,
                                PlacesAdminService.COMP_NAME };

        m_session.loadComponents( compNames );
        m_session.start();

    }
    catch(DuplicateObjectException e)
    {
        e.printStackTrace();
        exit();
    }

    // .... code removed
}
```

Once the session is started, we add a PlacesAdminServiceListener and a ActivityServiceListener so we will receive events from the PlaceAdminService and the ActivityService, both of which will be discussed later in this chapter.

Next, we use the code in Example 8-8 to log into the Sametime server as a server application.

Example 8-8 Logging in as a server application

```
void login(String host)
{
    m_saService =
        (ServerAppService)m_session.getCompApi(ServerAppService.COMP_NAME);

    short loginType = STUserInstance.LT_SERVER_APP;
    int[] supportedServices = {
        RedPanelDiscussionConstants.ACTIVITY_TYPE };

    // Server applications login directly to the server, and not through
    // the mux. So we can't use the default port.
    Connection[] connections = {
        new SocketConnection(1516, 17000), };
    m_saService.setConnectivity(connections);

    m_saService.addLoginListener(this);
}
```

```
        m_saService.loginAsServerApp(  
            host, loginType, RedPanelDiscussionConstants.PLACE_DISPLAY_NAME,  
            supportedServices);  
  
    }
```

Before calling the loginAsServerApp() method, we establish connectivity to the Sametime server by creating a new SocketConnection object and then calling the setConnectivity() method of the ServerAppService class. We have to make a direct socket connect to the Sametime server to avoid connecting through a MUX. The socket connection is opened on port 1516, which is the port Sametime listens on. The second parameter is the connection time-out setting. After we establish connectivity, we call the loginAsServerApp() method. The method takes four input parameters. These parameters and their descriptions are listed in Table 8-2.

Table 8-2 loginAsServerApp() parameters

Parameter	Purpose	Data type
host	DNS name of the server to connect to.	String
loginType	The type of login. In this example, we want to log in as a server application, so we use STUserInstance.LT_SERVER_APP.	int
appName	The application name.	String
serviceTypes	List of service types that the server application supports. Examples of service types are chat, audio, video, and so on. In this example, we are only concerned with the custom activity registering so we use RedPanelDiscussionConstants.ACTIVITY_TYPE.	int[]

Important: The serviceTypes parameter is where you declare the list of services that the server application will provide. In this application, we provide an Activity, so we have include the activity type number in the list.

If the log is successful, then the loggedIn() event will be triggered. We do not take any action in this event except to write a message to the console.

Figure 8-4 on page 206 summarizes the sequence of events when logging in as a server application.

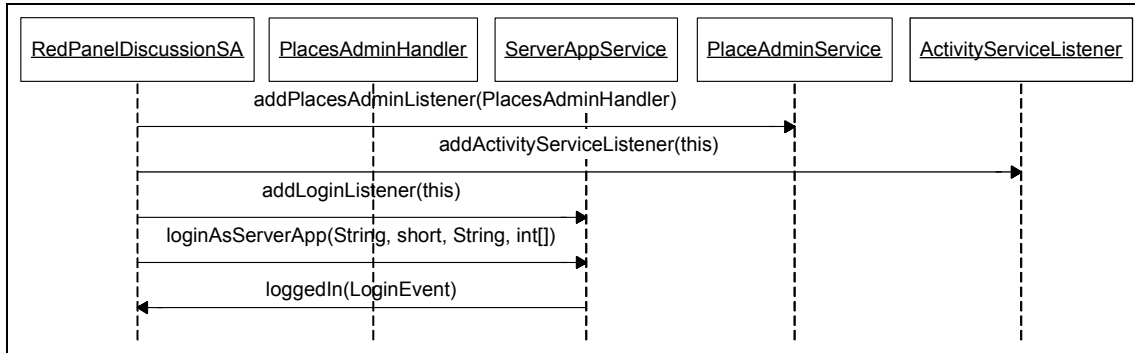


Figure 8-4 Login sequence diagram

8.5.2 Creating a persistent Place and Place characteristics

If the login is successful and the Places Admin Service is available, then the `serviceAvailable` event in the `PlaceAdminHandler` class will be triggered. Once this event is received, the application starts the process of defining some of the characteristics of the Place type that our Place will be created with. The first characteristic we set is the default number of sections in the Place type:

```

public void serviceAvailable(PlacesAdminEvent event) {
    m_adminService.setDefaultSections(
        RedPanelDiscussionConstants.PLACE_TYPE,
        RedPanelDiscussionConstants.NUM_SECTIONS,
        RedPanelDiscussionConstants.MAX_USERS_IN_SECTION);
}

```

The first parameter in the method is the `placeType`. The concept of Place type is something that seems to be glossed over in most discussions. A Place type, in a nutshell, is a template for a Place which can have preset characteristics, such as the default number of sections, a default activity, and other activities. In our example, the Place type gets created when we call this method. The second parameter is the number of sections to include in the Place. In the example, we need six sections (one Lobby, one Queue, and four Audience sections). By default, Sametime will always create one section for the stage, so even though we want seven sections in total, we only request six sections in this method call. The final parameter is the maximum number of users allowed in this section. We arbitrarily chose 30. Setting the value equal to 0 allows an unlimited number of users.

If the `setDefaultSections()` call is successful, then the `defaultSectionsSet()` event is triggered. In this event, we set the default activity for this Place type to the activity we defined when we logged in:

```
public void defaultSectionsSet(PlacesAdminEvent event) {
    m_adminService.setDefaultActivity(
        RedPanelDiscussionConstants.PLACE_TYPE,
        RedPanelDiscussionConstants.ACTIVITY_TYPE,
        null);
}
```

If the `setDefaultActivity()` call was successful, then the `defaultActivitySet()` event is triggered. In this event, we create our persistent Place using the `createPersistentPlace()` method:

```
public void defaultActivitySet(PlacesAdminEvent event) {
    m_adminService.createPersistentPlace(
        RedPanelDiscussionConstants.PLACE_NAME,
        RedPanelDiscussionConstants.PLACE_DISPLAY_NAME,
        RedPanelDiscussionConstants.PLACE_TYPE,
        "",
        EncLevel.ENC_LEVEL_NONE);
}
```

In this method call, we pass the internal and display name of the Place and the Place type number of the Place type we just created.

If the `createPersistentPlace()` method call is successful, the `placeCreated()` event is triggered. No action is taken when this event is received, except to write a message to the console. However, since the Place has been created and we added a default activity to the Place, the `activityRequested()` event is triggered.

Figure 8-5 on page 208 summarizes the process for creating a persistent Place and setting its default characteristics.

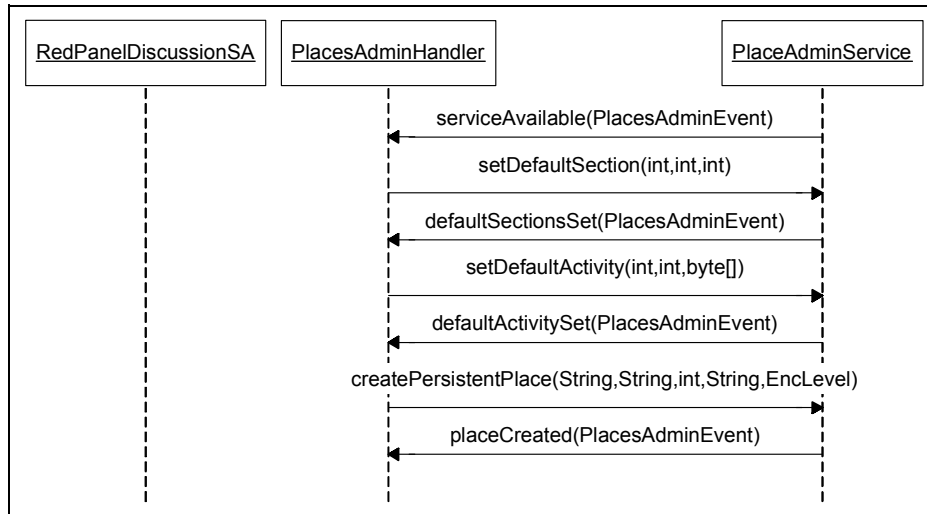


Figure 8-5 Creating persistent Place and characteristics sequence diagram

This leads up to our next section on responding to a request for the activity.

8.5.3 Responding to a request for the activity

Because we set the default activity for the Place to be our custom activity, the `activityRequested()` event will be fired after the Panel Discussion Place has been created. When we receive this event, we check to ensure that the Place requesting the activity was created with our Place type. If so, we accept the activity and instantiate a new `ActivityHandler` class. The following is the code we used for handling the request for the activity:

```

public void activityRequested(ActivityResult event) {
    MyActivity myActivity = event.getMyActivity();
    if (myActivity.getPlace().getType() ==
        RedPanelDiscussionConstants.PLACE_TYPE)
    {
        m_activityService.acceptActivity(myActivity, null);
        new ActivityHandler(event.getPlace(), myActivity);
    } else {
        m_activityService.declineActivity(myActivity, 0);
    }
}

```

Note: Even though in the panel discussion application we only create one Place, the server application does have the ability to service requests for the activity from any number of Places. The server application needs to serve as a type of request dispatcher. Therefore, it is a good practice to instantiate a new class for each activity request.

The ActivityHandler class is responsible for listening for incoming text messages, translating them, and then distributing them to the different sections. It also handles some program control related data message.

Figure 8-6 depicts the process of accepting the activity request.

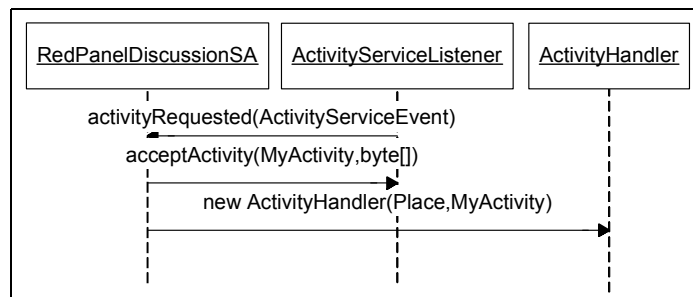


Figure 8-6 Accepting the custom activity request

8.5.4 Monitoring the Place

When the ActivityHandler object is created, it in turn instantiates a PlaceMonitor class. The PlaceMonitor class manages the different sections within the Place. It listens for new sections being added to the Place. As each section is added, it uses an algorithm to determine which section is the stage, which section is the Lobby, and which section is the Queue. The stage section is a special section in the eyes of the Sametime Places model. The stage section can be determined by using the isStage() method. The Lobby section is determined by finding the non-stage section with the lowest Member ID. By default, all users entering a Place will enter in this section unless a flag was set programmatically to enter the stage. The Queue section is defined as the section with the lowest Member ID that is not the stage section and not the Lobby section. Example 8-1 on page 196 shows the algorithm we used.

```
public void sectionAdded(PlaceEvent event) {
    Section section = event.getSection();
    Section tempSection = null;

    if (section.isStage()) {
        System.out.println("Event: section added " +
section.getMemberId().toString() + " (stage)");
        m_stage = section;
        if (m_sections.containsKey(section.getMemberId()) == false) {
            m_sections.put(section.getMemberId(), section);
        }
    } else if (m_lobby == null) {
        System.out.println("Event: section added " +
section.getMemberId().toString() + " (lobby)");
        m_lobby = section;
        m_lobby.addSectionListener(m_lobbyMonitor = new LobbyMonitor(this));
        m_lobby.putAttribute(new
STATtribute(RedPanelDiscussionConstants.LOBBY_KEY, ""));

        } else {
            if (section.getMemberId().intValue() <
m_lobby.getMemberId().intValue())
            {
                System.out.println("Action: swapping section and lobby" +
section.getMemberId().toString());
                System.out.println("Debug: old lobby " +
m_lobby.getMemberId().toString());
                tempSection = section;
                section = m_lobby;
                m_lobby.removeAttribute(RedPanelDiscussionConstants.LOBBY_KEY);
                m_lobby.removeSectionListener(m_lobbyMonitor);

                m_lobby = tempSection;
                m_lobby.putAttribute(new
STATtribute(RedPanelDiscussionConstants.LOBBY_KEY, ""));
                tempSection = null;
                System.out.println("Debug: new lobby " +
m_lobby.getMemberId().toString());
            }

            if ((m_sections.isEmpty() || m_sections.size() <= 1) && m_queue ==
null) {
                // this section will be the ask a question queue
                System.out.println("Event: section added " +
section.getMemberId().toString() + " (queue)");
                m_queue = section;
            }
        }
    }
}
```

```

        m_queue.addSectionListener(m_queueMonitor = new QueueMonitor(this,
m_queue));
        m_queue.putAttribute(new
STAttribute(RedPanelDiscussionConstants.QUEUE_KEY, "OK"));
// temp stop
//
        m_sections.put(m_queue.getMemberId(), m_queue);

    } else {
        System.out.println("Event: section added " +
section.getMemberId().toString() + " (regular)");
        if (m_sections.containsKey(section.getMemberId()) == false) {
            m_sections.put(section.getMemberId(), section);
            SHandler sh;
            section.addSectionListener(sh = new SHandler());
            section.putAttribute(new
STAttribute(RedPanelDiscussionConstants.SECTION_NAMES_KEY,
RedPanelDiscussionConstants.SECTION_NAMES[m_ndx]));
            section.putAttribute(new
STAttribute(RedPanelDiscussionConstants.SECTION_LANGUAGES_KEY,
RedPanelDiscussionConstants.SECTION_LANGUAGES[m_ndx]));
            section.removeSectionListener(sh) ;
            // needed to add a listener for failed attributes
            m_ndx++;
        }
    }
}
}
}
}

```

The LobbyMonitor class monitors users entering the Lobby section. When a user enters the section, it tries to move them into the appropriate section based on a properties file.

When the Queue section is added, a QueueMonitor class is instantiated, which monitors users entering and leaving the Queue section. This class will be discussed further in the next chapter.

Figure 8-7 on page 212 shows the process for setting up the different Place and section monitors.

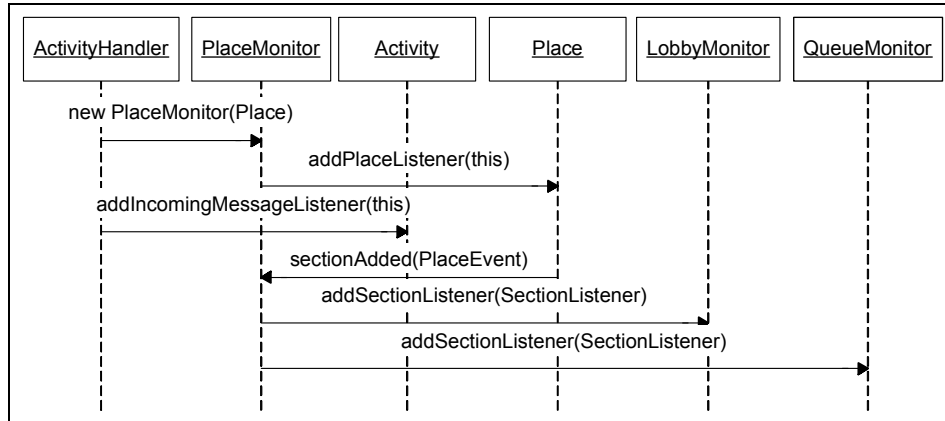


Figure 8-7 Process for setting up Place and section monitors

8.5.5 Managing the Queue

The QueueMonitor class is responsible for monitoring users entering and leaving the Queue section. When a user enters the Queue section, we perform some housekeeping chores. We store their UserInPlace object in a hashtable so that we have a means of communicating with the user when we need to. We then add their Member ID to a vector, which serves as the Queue for asking questions to the panel. Users are added in a first-in-first-out basis. Finally, we add a UserInPlaceListener for the user. We needed to add this listener in order to access the users attributes; otherwise, we could not access them using the UserInPlace.getAttributes() method.

When a user enters the Queue, the userEntered() event is fired. If the Queue vector is empty when a user enters the section, the user is given permission to ask the panel a question. We use the UserInPlace.sendData() method to notify the user that it is their turn. The user then has the right to speak until he is finished. When finished, the user will clicks a button in the client application, which will move them back into the Lobby. When they leave the section, the userLeft() event is triggered, which removes the user from the hashtable and Queue, removes the listener, and then notifies the next user.

Example 8-10 on page 213 shows the code used to monitor users entered and leaving the Queue section.

```
public void usersEntered(SectionEvent event) {
    UserInPlace[] users = event.getEnteredUsers();

    for (int i = 0; i < users.length; i++) {
        m_usersInPlace.put(users[i].getMemberId(), users[i]);
        m_userQueue.add(users[i].getMemberId());
        NoActionUserInPlaceListener nal = null;
        users[i].addUserInPlaceListener(nal = new
NoActionUserInPlaceListener());
        m_userListeners.put(users[i].getMemberId(), nal );

        if (m_userQueue.size() == 1) {
            this.notifyMember(users[0].getMemberId());
        }
    }
}

public void userLeft(SectionEvent event) {
    UserInPlace userInPlace = event.getDepartedUser();
    m_usersInPlace.remove(userInPlace.getMemberId());
    m_userListeners.remove(userInPlace.getMemberId());
    if (m_userQueue.elementAt(0) == userInPlace.getMemberId()) {
        m_userQueue.remove(0);
        if (m_userQueue.size() >= 1) {
            this.notifyMember((Integer) m_userQueue.elementAt(0));
        }
    } else {
        m_userQueue.remove(userInPlace.getMemberId());
    }
}
```

8.5.6 Receiving, translating, and responding to questions

When a user in the Queue sends a question to the panel, they are actually sending a message to the custom activity in our server application. The message will trigger the `textReceived()` event the `ActivityHandler` class. The code first get the senders display name and language from their user attributes. Next, it gets a list of all the country sections. (The list does not include the stage, Lobby or Queue.) For each section, it grabs the defined language for the section, then translates it and sends it to the section. Then it performs the same process for all users in the Queue. Finally, it translates the message into English and sends it to the panel section.

Example 8-11 on page 214 shows the code used in the `textReceived()` event.

```
public void textReceived(MessageEvent event) {
    Enumeration attribs;
    STAttribute attrib;
    Enumeration sections;
    Section section;
    String senderName = "";
    String inputText = "";
    String translatedText = "";
    String senderLanguage = "EN";
    String receiverLanguage = "EN";
    String hashedText = "";
    Hashtable translatedMsgs = new Hashtable();

    sections = m_watchedPlace.getSections();

    // read input data
    Enumeration senderAttribs = event.getSender().getAttributes();
    STAttribute senderAttrib;
    while (senderAttribs.hasMoreElements()) {
        senderAttrib = (STAttribute) senderAttribs.nextElement();
        if (senderAttrib.getKey() ==
RedPanelDiscussionConstants.SECTION_LANGUAGES_KEY) {

            senderLanguage = senderAttrib.getString();
        } else if (senderAttrib.getKey() ==
RedPanelDiscussionConstants.MY_DISPLAY_NAME_KEY) {

            senderName = senderAttrib.getString();
        }
    }
    inputText = event.getText();

    // send message to each section in its language
    while (sections.hasMoreElements()) {
        section = (Section) sections.nextElement();
        // add part to get language
        attribs = section.getAttributes();
        while (attribs.hasMoreElements()) {
            attrib = (STAttribute) attribs.nextElement();
            if (attrib.getKey() ==
RedPanelDiscussionConstants.SECTION_LANGUAGES_KEY) {
                try {
                    receiverLanguage = (attrib.getString().equals("")) ?
receiverLanguage : attrib.getString();
```



```

        hashedText = (String) translatedMsgs.get(senderLanguage
+ " _" + receiverLanguage);
        if ( hashedText == null) {
            if (senderLanguage.equals(receiverLanguage) == false
) {
                translatedText = this.translate(inputText,
senderLanguage, receiverLanguage);
            } else {
                translatedText = inputText;
            }
            translatedMsgs.put(senderLanguage + " _" +
receiverLanguage, translatedText);
        } else {
            translatedText = hashedText;
        }
        section.sendText(senderName + "      " + translatedText);
    } catch (Exception e) {

        section.sendText(senderName + "      " + event.getText()
+ " (translation error)");
    }
    break;
}
}
}

// send text to all users in the queue in their language
Enumeration qmems = m_watchedPlace.getQueue().getUsers();
UserInPlace qmem = null;
Enumeration qmem_attribs;
STAttribute qmem_attrib = null;

while (qmems.hasMoreElements()) {
    qmem = (UserInPlace) qmems.nextElement();
    qmem_attribs = qmem.getAttributes();
    while(qmem_attribs.hasMoreElements()) {
        qmem_attrib = (STAttribute) qmem_attribs.nextElement();

        if (qmem_attrib.getKey() ==
RedPanelDiscussionConstants.SECTION_LANGUAGES_KEY) {
            try {
                receiverLanguage = (qmem_attrib.getString().equals("")) ?
receiverLanguage : qmem_attrib.getString();
                hashedText = (String) translatedMsgs.get(senderLanguage +
" _" + receiverLanguage);
                if ( hashedText == null) {
                    if (senderLanguage.equals(receiverLanguage) == false ) {
                        translatedText = this.translate(inputText,
senderLanguage, receiverLanguage);

```

```

        } else {
            translatedText = inputText;
        }

        translatedText = this.translate(inputText,
senderLanguage, receiverLanguage);
        translatedMsgs.put(senderLanguage + " " +
receiverLanguage, translatedText);
        } else {
            translatedText = hashedText;
        }

        qmem.sendText(senderName + " " + translatedText);
    } catch (Exception e) {
        qmem.sendText(senderName + " " + inputText + "
(translation error)");
    }
}
}

// send to the panel section in english
try {
    translatedText = this.translate(inputText, senderLanguage, "EN");
    hashedText = (String) translatedMsgs.get(senderLanguage + " " +
receiverLanguage);
    if ( hashedText == null) {
        if (senderLanguage.equals(receiverLanguage) == false ) {
            translatedText = this.translate(inputText, senderLanguage,
receiverLanguage);
        } else {
            translatedText = inputText;
        }

        m_watchedPlace.getStage().sendText(senderName + " " +
translatedText);
        // no need to cache because we are done after this
    } else {
        translatedText = hashedText;
    }

    m_watchedPlace.getStage().sendText(senderName + " " +
translatedText);
    } catch (Exception e) {

        m_watchedPlace.getStage().sendText(senderName + " " + inputText +
"(translation error)");
    }
}

```

}

8.6 Summary

In this chapter, we have discussed both the client- and server-side of an application that uses the Sametime Places model and the Sametime Community Server Toolkit. We hope this chapter has demonstrated several of features of the different Java Toolkits and provided another example of how to leverage the Sametime Places architecture.



Sametime Links

Sametime Links is a lightweight toolkit that allows Web developers to Sametime-enable their Web pages and applications with “live names.” While it has a simple API, it has a rich feature set. It includes functionality for providing awareness with status icons, instant messaging and n-way chats, launching instant meetings, setting status, and supporting events within Sametime Virtual Places. Since the user interface is HTML, it is fully customizable.

Within this chapter, we will provide an overview of the Sametime Links Toolkit and review the basics of preparing a page for Sametime Links “live names”. Next, we explain the usage and purpose of each file that is used by Sametime Links, and extend that to an example of using Sametime Links on a Web site to build in visitor interaction. This chapter will cover the following topics:

- ▶ Basic review for preparing a an HTML or JSP page for enabling live names
- ▶ How to provide customized/branded ST Links experience for visitors
- ▶ Allowing an internal agent to see who is on the Web site
- ▶ Allowing an internal agent to see which page the visitor is on
- ▶ Allowing an internal agent to interact with the visitor on the Web site
- ▶ Let internal agents looking at visitors on the page see that they are already chatting with another agent
- ▶ Reviewing how to customize pop-up menu options from Sametime Links

9.1 Overview of Sametime Links

The Sametime Links document describes Sametime Links (ST Links) as “a lightweight toolkit that allows Web developers to Sametime-enable their Web pages and applications with live names. A simple HTML/JavaScript API allows Web developers to turn existing names into Sametime Links by simply adding a few lines of HTML, without affecting the layout of the page. While rich in functionality, it is light in size, using an embedded applet of only about 20K.”

Please refer to the *Sametime Links 3.1 Toolkit Developer's Guide and Reference* (see 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22 for information on finding this guide) for a complete review of functionality, Sametime Links elements, and advanced features and options for customization. This document is accessible from the Sametime 3.1 Toolkit page. You can go to the Toolkit home page by selecting the SDK link in the Sametime server home page. In the Toolkits page, select the Sametime Links Toolkit link. If there is no link, then you first have to install the Sametime Toolkits package on the Sametime server. You can also find the Developer's Guide and the toolkit download on the Toolkits and Drivers page.

9.2 Deployment considerations

When deploying Sametime Links within an organization, there are certain considerations that should be taken into account, such as:

- ▶ Does the Sametime Server version you are running support ST Links?
- ▶ Does your Sametime Server allow external HTTP connections?
- ▶ What impact will the downloading of the ST Links applet have on your server?
- ▶ Will you allow anonymous access to your Sametime Server for ST Links users?
- ▶ What browsers do you need to support?

9.2.1 Size is important

Size, or lack of it, is one of the greatest benefits of ST Links over the regular Sametime Java Toolkits. Sametime Links tends to be much smaller in size than most of the applications that have been built with the Sametime Java Toolkits. The reason for this is that much of the functionality within ST Links is actually executing and residing on the Sametime Server, with the small ST Links applet primarily acting as a medium between the client and server, doing things like encrypting/decrypting chat conversations, handling the display of users status

and messages, and so on. ST Links is somewhere between 25 KB and 35 KB in size, depending upon the client Java Virtual Machine being used.

The majority of applications that have been using the Sametime Java Toolkits often include the entire STComm.jar and STRes.jar files, which results in a download of about 450 KB, so it is easy to see the benefits of Sametime Links from a size point of view. The chart in Figure 9-1 shows how that difference can impact your Web site network traffic for a single day when using an application built around ST Links vs. using the Sametime Java Toolkit.

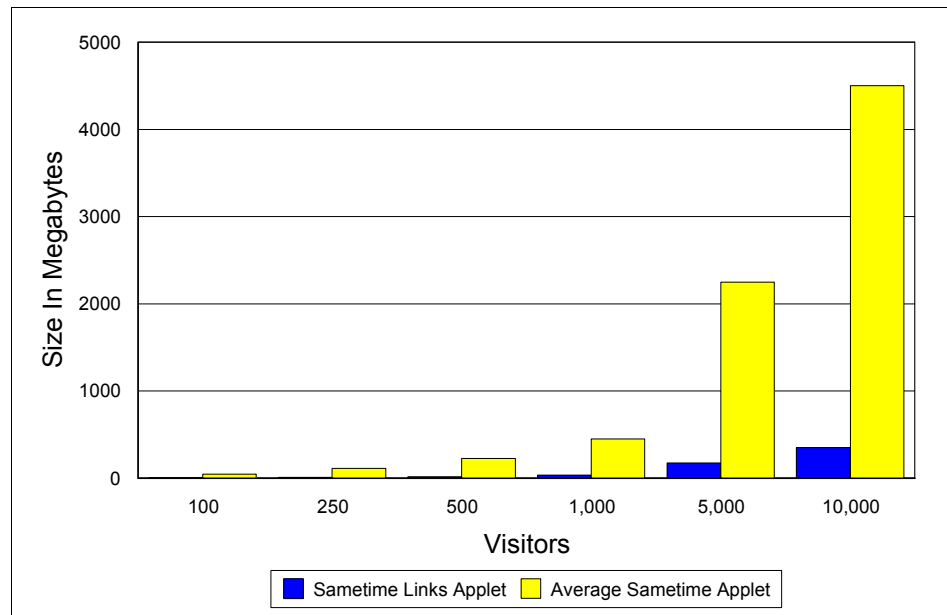


Figure 9-1 Sametime Links vs. average Sametime applet download size

The size of the ST Links Toolkit has the biggest impact on visitors to your page the first time they load it. The ST Links applet is downloaded roughly 12 times quicker than a page using the regular Sametime Java Toolkit.

9.2.2 Platform support

The IBM Lotus Sametime Links Toolkit 3.1 is targeted for use with the Sametime Server 2.5 and above. Using the toolkit with a Sametime Server 2.5 requires a server add-on, which is available from IBM.

If you are running Sametime 3.0, it is highly recommended that you update your installation to Sametime 3.0 with Interim Fix 2, as this addresses a number of issues that can arise when using Sametime Places within Sametime Links and

Lotus QuickPlace. Contact IBM Lotus support for details on how to obtain Interim Fix 2. If you have not installed Interim Fix 2 and start to use Sametime Places within ST Links, you will experience issues like the ST MUX continually shutting down and restarting.

STLinks Toolkit now has multiplatform support in Sametime 3.1. Platform support now includes AIX, Solaris, and Linux platforms. Browser support now also includes Netscape 7 instead of Netscape 4.7. Operating system and browser compatibility is shown in Table 9-1.

Table 9-1 Operating system and browser compatibility for Sametime Links Toolkit

Operating system	Browser version
Windows	IE 5 and higher, Netscape 7
AIX	Netscape 7
Solaris	Netscape 7
Linux	Netscape 7

9.2.3 Working with anonymous users

Logging in as anonymous lets anonymous users log in to the community. Anonymous login to a Sametime Community must be enabled by the administrator. To log in as anonymous, pass an empty string as the key argument to writeSTLinksApplet. If you pass an empty string as the loginName argument, the server assigns a name with the form "UserN/Guest", where N is a number. If you pass a name as the loginName argument, the name that the server assigns is "name/Guest". The prefix "User" and the suffix "Guest" can be changed by the administrator.

Note: Additional permissions for anonymous users might be required. For example, to allow Sametime Links to resolve names, the administrator should turn on the anonymous access setting "Users can type names (resolve users and groups) to add them to an awareness list." This setting is not required if you provide Sametime Links with resolved names (bResolve set to false in the writeSametimeLink call). For Place-based awareness, the administrator should turn on the anonymous access setting "Users can participate in meetings or enter virtual Places."

9.3 Enabling live names in a Web page

The capability to enable live names from within an HTML page or JSP page is one of the most basic functions provided by the STLinks Toolkit and is thoroughly documented in the *Sametime Links 3.1 Toolkit Developer's Guide and Reference*. We are including these steps as a basic review, and to ensure thorough coverage of using ST Links. Please refer to the *Sametime Links Toolkit Developer's Guide and Reference* for a sample HTML page that illustrates this concept in greater detail.

The following three steps outline the preparation for including Sametime Links into an HTML or JSP page:

1. We begin by inserting the code to set required parameters to call Sametime Links in the HEAD tag of either our HTML Page or JSP page.

Note: The HEAD tag of a JSP page is exactly the same as any HTML page.

Hence, we need to add the code shown in Example 9-1.

Example 9-1 HTML HEAD tag to call Sametime Links

```
<HEAD>
...
<LINK REL=STYLESHEET HREF="http://<st server>/sametime/stlinks/stlinks.css"
TYPE="text/css">

<SCRIPT src="http://<st server>/sametime/stlinks/stlinks.js"></SCRIPT>

<SCRIPT>
    setSTLinksURL("http://<st server>/sametime/stlinks","en",
    "http://<st server>/sametime/stlinks")
</script>
...
</HEAD>
```

Note that the *<st server>* parameter refers to the proper, fully qualified host name of your Sametime server.

2. Next, we call the Sametime Links applet by passing the necessary login information, as shown in Example 9-2 on page 224.

Example 9-2 Call to the Sametime Links Applet with logon information

```
<script>
  writeSTLinksApplet("<user name>", "<password>", <is by token flag>);
</script>
```

By integrating your Web application server, the user name and password properties can be passed as a token. Without this token, the user would need to pass the login information again.

3. Finally, for each field where user names are shown, we included the `writeSametimeLink()` function, as shown in Example 9-3.

Example 9-3 writeSametimeLink() function

```
<script>
writeSametimeLink("<user id>", "<display name>", <resolve flag>, "<options>");
</script>
```

Once these modifications are saved within the HTML or JSP page and the proper Sametime server name is entered for the `<st server>` parameter, all names that appear within a Web page will be “live enabled” when a user logs on.

9.4 Sametime Links directory overview

When planning to customize Sametime Links, the best approach to understanding it is to review and understand the contents of the Sametime Links directory that the install creates. We recommend this approach over simply trying to focus on the JavaScript function calls that are available within the product. All the Sametime Links dialogs are HTML pages. This allows you to easily customize the graphics; for example, you can change the background color or add your company's logo. You can also add functionality to the dialogs or disable existing functionality. In the next few pages, we will go through the contents of these directories and files. By understanding the contents of these directories, it will make life much easier when you come to customize these pages.

9.4.1 Directory contents

The Sametime Links directory (STLINKS) can be found within a subdirectory of the Sametime Servers data directory, typically in the location `serverdatadirectory/domino/html/sametime/stlinks`.

Within this directory, you will find a number of HTML files and directories. The directories fall into two categories:

- ▶ Language support files: allLang, and the two letter directories, for example, en, de, and es; if you are customizing the HTML files, and are supporting multiple languages, you will need to modify the HTML contents of each directory. The HTML files will be covered in more detail in a later section.
- ▶ Image Directory: img; this directory contains all the graphics used within Sametime Links (see Figure 9-2 for an overview of the graphic image).

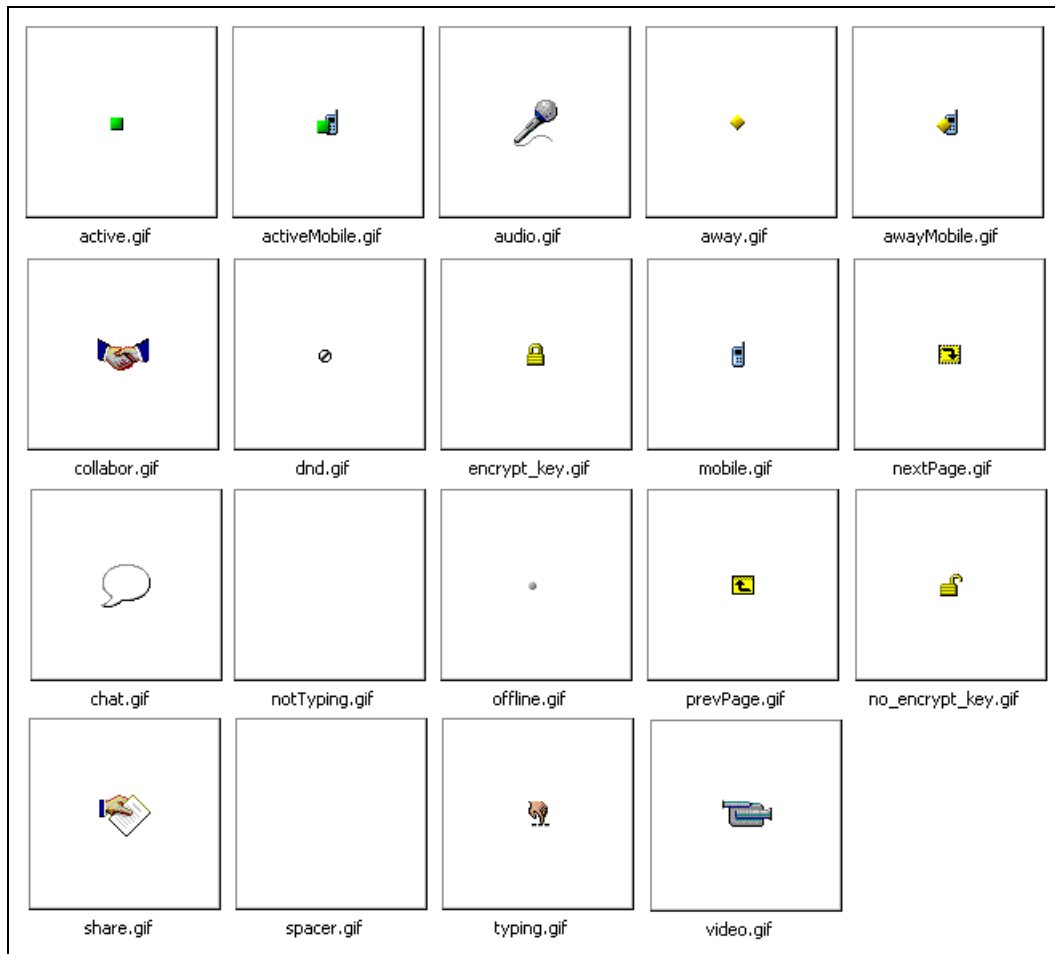


Figure 9-2 Sametime Links graphic images

The files within the STLINKS directory are:

- ▶ `Stlinks.js`: Contains the JavaScript functions used within the HTML pages to interact with the applet, and to launch the various interaction windows.
- ▶ `stlinks.cab` and `stlinks.jar`: The applet files for the MS JVM and SUN JVM, respectively.
- ▶ `DebugLevel.class`: This file specifies a debug level to Sametime Links.
- ▶ `hostInfo.js`: Contains information concerning the tunneling details for Sametime 2.5 servers.
- ▶ `stlinks.css`: Contains the default cascading style sheet used for Sametime Links pages. This page can be edited with any third-party CSS editors, or even with Notepad. This file, when used in combination with the contents of the `img` directory, can be used to very quickly alter the appearance of the Sametime Link names within a page. By making some simple changes to the `img` files in the CSS file, the normally green Sametime Links names can suddenly take on an appearance that is similar to the rest of your Web site.

9.4.2 Understanding the HTML files

The HTML files that are contained within the various language subdirectories and the `allLang` directory are the key to being able to change the appearance of the Sametime Links dialog boxes. In this section, we will outline the functions of each file and their location.

Files within the `allLang` subdirectory

The files within the `allLang` subdirectory are common across all the languages. Changes to a file in this directory will appear within all client languages.

borderFrame.html

This file contains a grey background, which acts as the border in many dialogs, but also acts as a place holder to receive text through `document.write` commands from the Sametime Links applet and other functions within numerous pages.

`borderFrame.html` is referenced in:

- ▶ `announcement.html`
- ▶ `im.html`
- ▶ `invitation.html`
- ▶ `inviteOthers.html`
- ▶ `nway.html`
- ▶ `place.html`
- ▶ `placeChat.html`

statusFrame.html

This file is used to construct the status bar at the bottom of many of the dialogs. It is also used via a parameter to display the encryption icon in the bottom right of many dialogs, as shown in Figure 9-3.

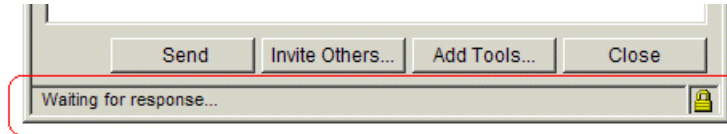


Figure 9-3 *statusFrame.html*

StatusFrame.html is referenced in:

- ▶ announcement.html
- ▶ im.html
- ▶ invitation.html
- ▶ nway.html
- ▶ place.html
- ▶ placeChat.html

transcript.html

This file is used to display the chat history in various dialogs as the instant messaging conversation continues, as shown in Figure 9-4.

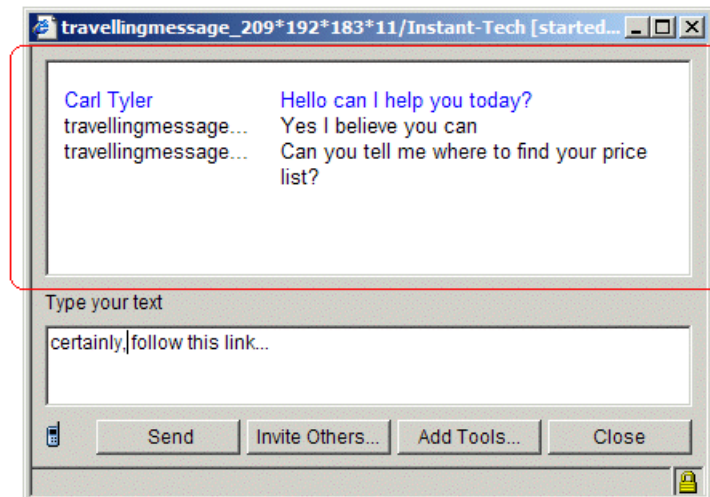


Figure 9-4 *transcript.html*

transcript.html is referenced in:

- ▶ im.html
- ▶ nway.html
- ▶ placeChat.html

Files within the individual language subdirectories

The files within the individual language subdirectories relate specifically to that country. If you are supporting multiple languages and change a file within one of these directories, you will need to change the same file in all the directories to be consistent across the languages. For the purpose of this redbook, we will be referring to the en (English) directory. The file names within the directories and their functions are the same for each country.

announceBtn.html

This file contains the Respond and Close button, used within the announceFrame.html, as seen in Figure 9-5.

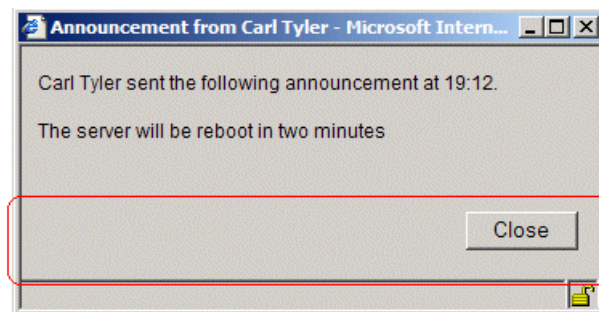


Figure 9-5 *announceBtn.html*

Figure 9-5 shows the Announcement window that appears when the person sending the announcement has not selected "Allow People to respond to me"; without that option being selected, the announcement would look like the window shown in Figure 9-6 on page 229, where we can see the Respond button is displayed.

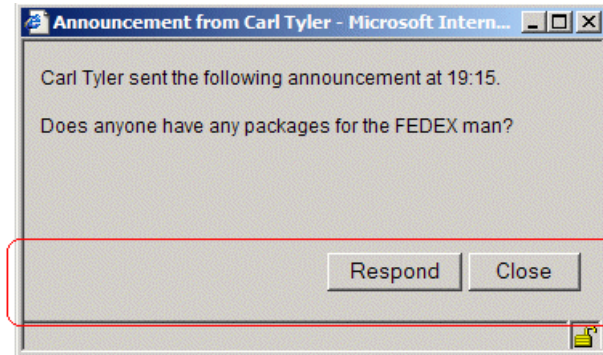


Figure 9-6 *announceBtn.html*

announceBtn.html is referenced in announcement.html.

announceFrame.html

This is the page that passes the announcement into the borderFrame.html contained within the announcement.html dialog (see Figure 9-7).

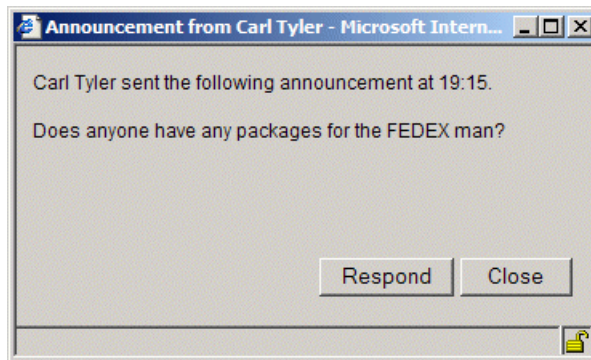


Figure 9-7 *announceFrame.html*

announceFrame.html is referenced in announcement.htm.

announcement.html

This dialog is displayed when an announcement is received. The frameset layout is shown in Figure 9-8 on page 230.

borderFrame.html (*x,*y), the contents of this frame are updated by announceFrame.html§	
announceBtn.html (*x,60y)§	
statusFrame.html (18x,*y)§	statusFrame?secureIcon(20x,18y)§

Figure 9-8 *announcement.html* frameset layout

The dialog box is shown in Figure 9-9.

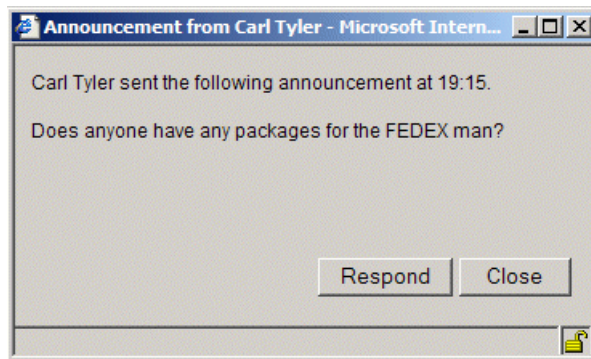


Figure 9-9 *announcement.html*

changeStatus.html

This dialog is displayed when the `openStatusWindow()` Sametime Links function is called (see Figure 9-10).

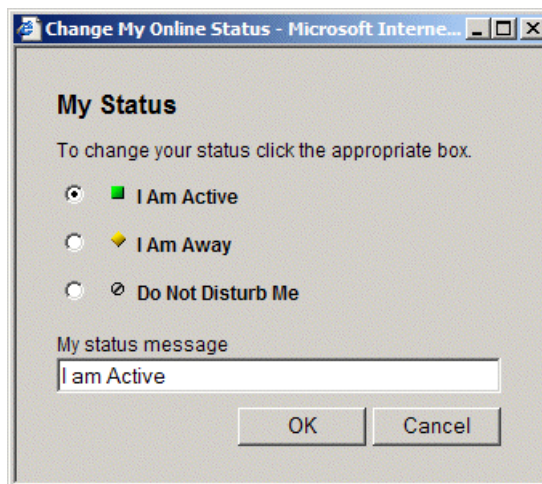


Figure 9-10 *changeStatus.html*

chatApplet.html

This page is never seen, but is used for all chat conversations. This hidden page contains the references to the chat functions of the Sametime Links applet. If you wish to do advanced customization of ST Links by modifying what gets sent or modifying what is received and so on. You will find most of the functions that you need to modify within this html file.

chatApplet.html is referenced in chatWindow.html.

chatBtn.html

This page holds the code to display the buttons Send, Invite Others..., Add Tools..., and Close (see Figure 9-11).

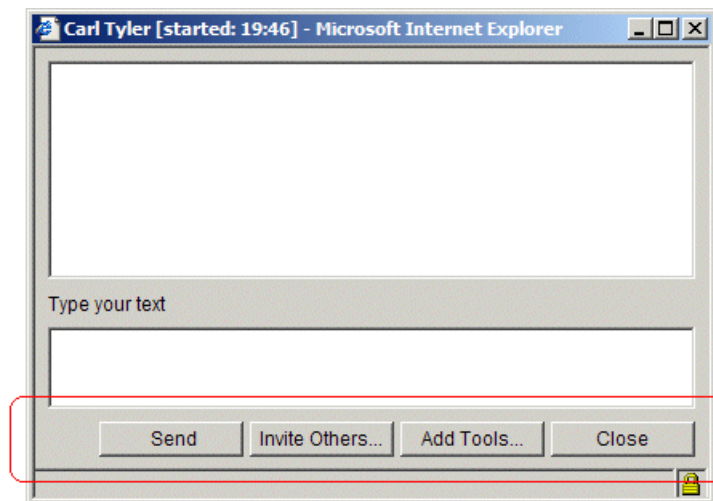


Figure 9-11 *chatBtn.html*

chatBtn.html is referenced in:

- ▶ im.html
- ▶ nway.html
- ▶ placeChat.html

chatWindow.html

This page is used to build the dialogs for:

- ▶ Place awareness lists/place.html
- ▶ Place Chat/placeChat.html
- ▶ IM Chat/im.html
- ▶ N-way chat/nway.html
- ▶ Invitation Dialog/invitation.html

The frameset layout is shown in Figure 9-12.

The contents of this frame are set via a parameter when launched by STLINKS.JS, the possible contents can be: 'place.html'¶ 'placeChat.html'¶ 'im.html'¶ 'nway.html'¶ 'invitation.html' (100%x,100%y)§
chatApplet.html (hidden: 0x,0y)§

Figure 9-12 chatWindow frameset layout

dirApplet.html

This page contains the functions for retrieving directory information when inviting users; it is not seen, as it is hidden in a frameset within inviteOthers.html

dirApplet.html is referenced in inviteOthers.html.

directory.html

This page contains the layout for the inviteOthers.html dialog. It shows the directory from the server, and allows users to be selected (see Figure 9-13 on page 233).

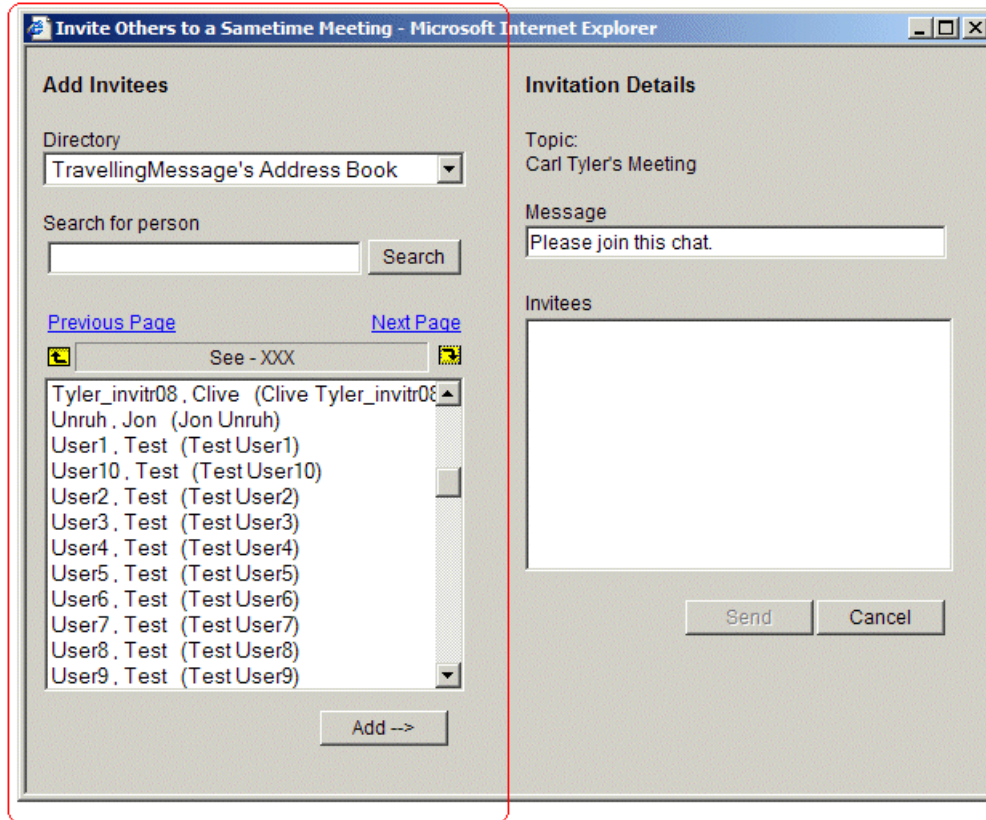


Figure 9-13 *directory.html*

directory.html is referenced in inviteOthers.html.

im.html

This page contains the layout for the instant messaging dialog when used in the chatWindow.html file (see Figure 9-14 on page 234).

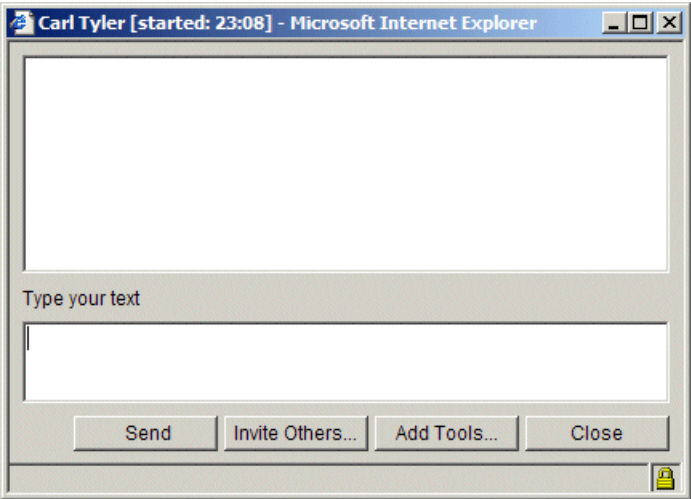


Figure 9-14 *im.html*

The frameset layout is shown in Figure 9-15.

borderFrame.html (*x,*y)\$		
borderFrame.html(8x,*y)\$	transcript.html (*x,*y)\$	borderFrame.html(8x,*y)\$
	inputFrame.html (*x,82y)\$	
	chatBtn.html (*x,36y)\$	
StatusFrame.html (*x,18y)\$		StatusFrame?secureicon (20x,18y)\$

Figure 9-15 *im.html frameset layout*

im.html is referenced in chatWindow.html.

inputFrame.html

This page (see Figure 9-16 on page 235) contains the chat entry box, and checks to see if you are hitting a key so that the other person can know if you are typing. The chat entry area is built by the function writeInputFrameApplet().

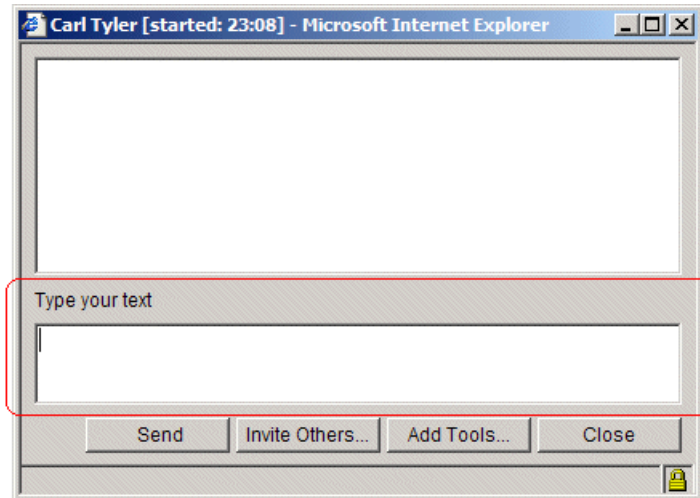


Figure 9-16 *inputFrame.html*

inputFrame.html is referenced in:

- ▶ im.html
- ▶ nway.html
- ▶ placeChat.html

invitation.html

This dialog (Figure 9-17) is displayed when someone is invited to a Meeting.



Figure 9-17 *invitation.html*

The frameset layout for invitation.html is shown in Figure 9-18 on page 236.

borderFrame.html ("x","y") The contents of this frame are replaced with inviteFrame.html§	
StatusFrame.html ("x,18y)§	StatusFrame?secureIcon (20x,18y)§

Figure 9-18 invitation.html frameset layout

invitees.html

This dialog (Figure 9-19) is displayed as part of the inviteOthers.html dialog when the **Invite others...** button is clicked.

Figure 9-19 invitees.html

invitees.html is referenced in inviteOthers.html.

inviteFrame.html

This dialog (Figure 9-20 on page 237) is placed into the invitation.html dialog when an invite is received.



Figure 9-20 *inviteFrame.html*

`inviteFrame.html` is referenced in `invitation.html`.

inviteOthers.html

This dialog (Figure 9-21 on page 238) displays the dialog for inviting other people into a chat or meeting.

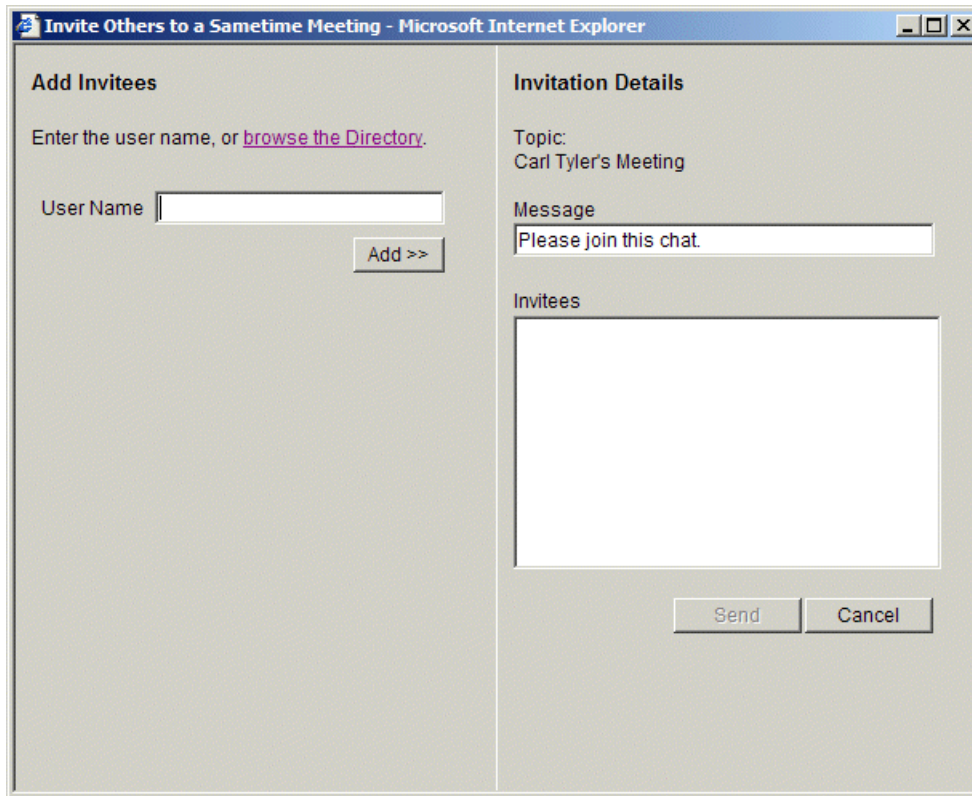


Figure 9-21 *inviteOthers.html*

The frameset layout for *inviteOthers.html* is shown in Figure 9-22.

resolve.html (50%x,100%y)\$	borderFrame.html (50%x,100%y)¶ The contents of this frame are replaced with invitees.html\$
dirApplet.html (hidden:0x,0y)\$	

Figure 9-22 *invieOthers.html* frameset layout

meetingTools.html

This dialog (Figure 9-23 on page 239) displays when adding tools to a chat.

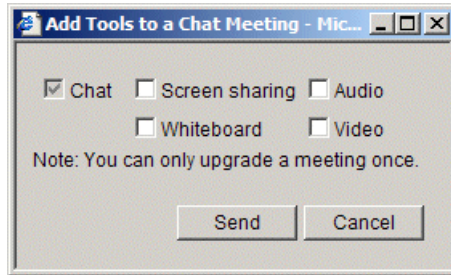


Figure 9-23 *meetingTools.html*

nway.html

This page (Figure 9-24) contains the layout for the n-way chat dialog when used in the chatWindow.html file.

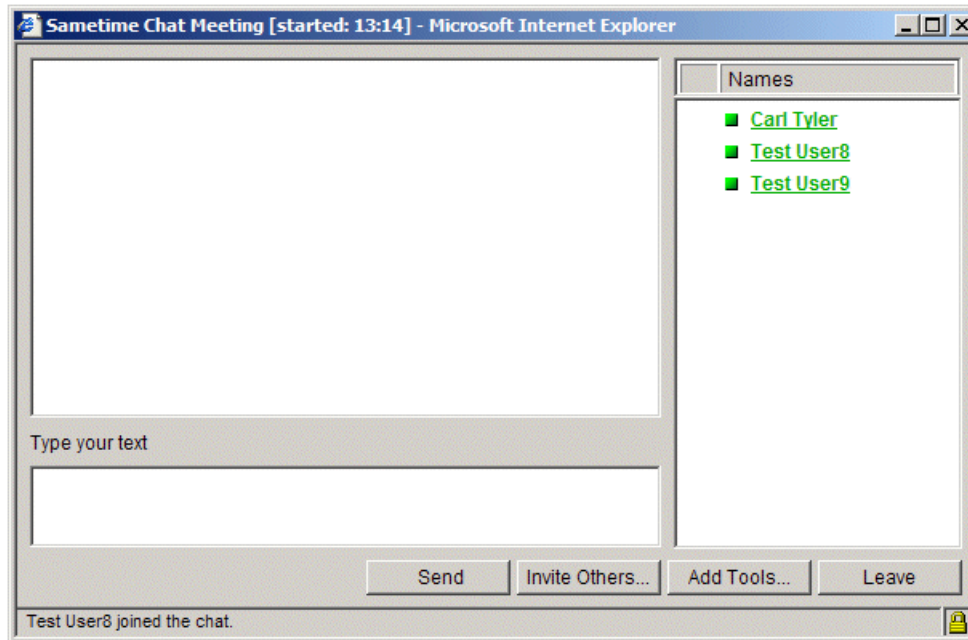


Figure 9-24 *nway.html*

The frameset layout for nway.html is shown in Figure 9-25.

BorderFrame.html (*x,8y)\$				
BorderFrame.html (8x,*y)\$	Transcript.html (*,*)\$	BorderFrame.html (8x,*y)\$	PeopleHeader.html (180x,25y)\$	BorderFrame.html (8x,*y)\$
	InputFrame.html (180x,82y)\$		PeopleList.html (180x,*y)\$	
	ChatBtn.html (*x,36y)\$			
StatusFrame.html (*x,18y)\$				StatusFrame.html?securelo n (20x,18y)\$

Figure 9-25 nway.html frameset layout

nway.html is referenced in chatWindow.html.

peopleHeader.html

This page (Figure 9-26) contains the heading for the awareness list within the n-way chat and placeChat dialogs.

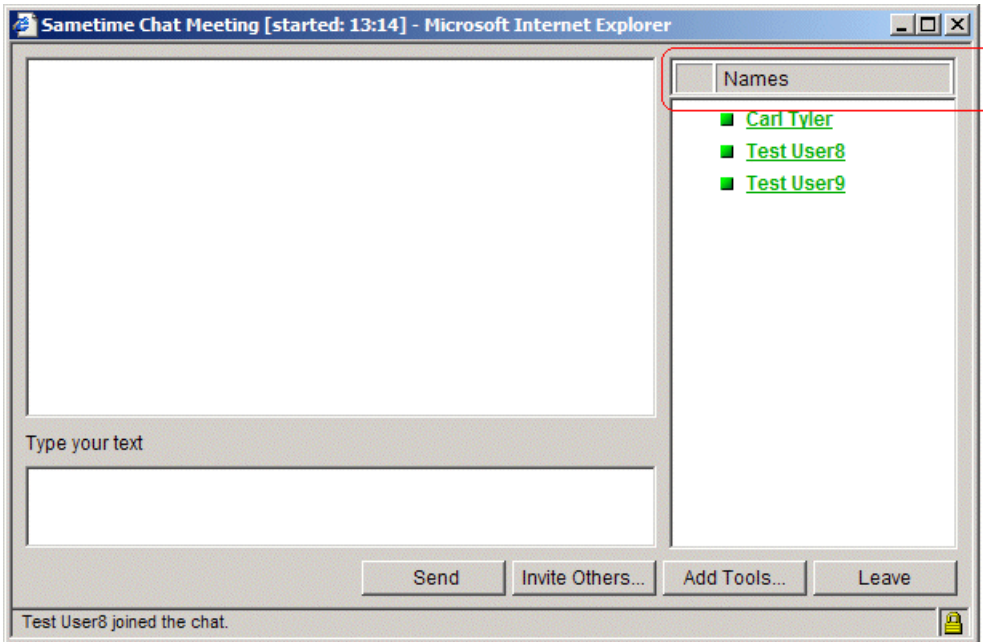


Figure 9-26 peopleHeader.html

peopleHeader.html is referenced in:

- ▶ nway.html
- ▶ placeChat.html

peoplelist.html

This page (Figure 9-27) contains the awareness list layout that is used in the place.html, nway.html and placeChat.html.

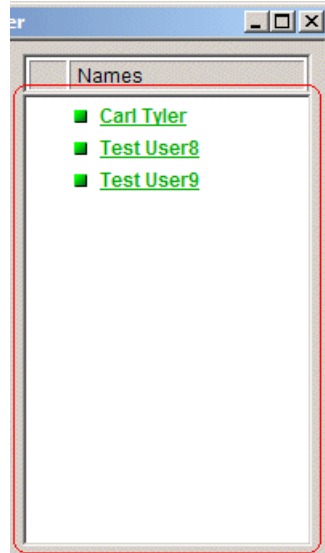


Figure 9-27 *peoplelist.html*

peopleList.html is referenced in:

- ▶ nway.html
- ▶ place.html
- ▶ placeChat.html

place.html

This page (Figure 9-28 on page 242) contains the layout for the Place Awareness list, which is displayed when the function openPlaceWin is executed, or a Sametime Links Place counter is clicked.

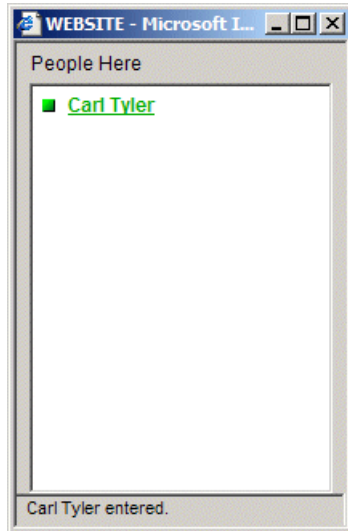


Figure 9-28 *place.html*

The frameset layout for *place.html* is shown in Figure 9-29.

BorderFrame.html (8x,25y)\$		
BorderFrame.html (8x,*y)\$	peopleList.html (*x,*y)\$	BorderFrame.html (8x,*y)\$
StatusFrame.html (8x,20y)\$		

Figure 9-29 *place.html* frameset layout

placeChat.html

This page (Figure 9-30 on page 243) contains the layout for the *placeChat* dialog when used in the *chatWindow.html* file. The layout appearance is practically identical to the *nway.html* layout (see earlier), except that *placeHeader.html* is used instead of *peopleHeader.html*.

placeChat.html is referenced in *chatWindow.html*.

placeHeader.html

This page displays the text "People Here" as the heading above the *peopleList* in the *placeChat.html*.

placeHeader.html is referenced in *placeChat.htm*.

resolve.html

This page (Figure 9-30) contains the layout for the Add Invitees within the inviteOthers.html dialog. It allows a person to type in the user name of a person to add or to click a button to browse the Directory; when the directory is browsed, this file is replaced with directory.html.

The screenshot shows a web browser window titled "Invite Others to a Sametime Meeting - Microsoft Internet Explorer". The main content area is divided into two vertical sections. The left section, titled "Add Invitees", contains the instruction "Enter the user name, or [browse the Directory](#)." Below this is a text input field labeled "User Name" and an "Add >>" button. The right section, titled "Invitation Details", contains a "Topic:" label with the value "Carl Tyler's Meeting", a "Message" label with a text input field containing "Please join this chat.", and an "Invitees" label with a large empty list box. At the bottom right of the right section are "Send" and "Cancel" buttons. A red rectangle highlights the "Add Invitees" section of the dialog.

Figure 9-30 *resolve.html*

resolve.html is referenced in inviteOthers.html.

res.js

Although res.js is not a HTML file, there is a copy stored in each of the language directories. It contains a number of resource strings for use within the other pages, and also contains the default x,y sizing for many of the dialogs. By editing this file, you can change the start-up size of many dialogs, and also change some of the strings presented to the user.

9.5 Building an interactive Web site

So now we know where the files are and what is in each of them, we can start building our interactive Web site. Let us go back to the original goals of our sample project:

- ▶ How to provide customized/branded ST Links experience for visitors
- ▶ Allowing an internal agent to see who is on the Web site
- ▶ Allowing an internal agent to see which page the visitor is on
- ▶ Allowing an internal agent to interact with the visitor on the Web site
- ▶ Let internal agents looking at visitors on the page see that they are already chatting with another agent

To make it easier to follow as we build this sample, we are going to call the internal person the "Agent" and the external visitor the "Customer"; in real life, the Customer could also be an internal user. As we will be modifying the UI for both the Agent and Customer to do different things, it is a good idea to have separate directories to manage them. So we are going to take the standard STLinks directory:

```
\Lotus\Domino\Data\domino\html\sametime\stlinks
```

and copy it and its contents twice to the new directories:

```
C:\Lotus\Domino\Data\domino\html\sametime\agentstlinks  
C:\Lotus\Domino\Data\domino\html\sametime\customerstlinks
```

For the changes we want to make to the agent side, we will edit the contents of the agentstlinks directory, and for customer changes, we will change the customerstlinks directory.

When loading Sametime Links in a Web page, we can now load the page that we want, so for customer facing pages, our Sametime Links code in the header would be:

```
<LINK REL=STYLESHEET HREF="<CODEBASE>/stlinks.css/" TYPE="text/css">  
<SCRIPT src="<CODEBASE>/stlinks.js">
```

Where <CODEBASE> is the URL of the directory where the ST Links run-time files are:

```
http://<STSERVERNAME>/sametime/customerstlinks
```

For Agent pages, the header would be the same, but the codebase would be:

```
http://<STSERVERNAME>/sametime/agentstlinks
```

9.5.1 Provide the customer with a branded ST Links experience

For this sample, we are going to complete some very simple customization on the screens that are displayed to the customer:

- ▶ Display a company logo in the top of the chat window
- ▶ Remove the buttons for Invite Others... and Add Tools....

Display a company logo in the top of the chat window

Referring to “im.html” on page 233, we can see that this file contains the instant messaging dialog layout. Looking at the layout of this HTML file more closely, we can see the easiest way to add a logo to this page is to edit the frameset, and make the top frames height larger and change the source file from borderFrame.html to something else that contains our logo; in this case, we will create a new file called logoFrame.html. This new file is the same as borderFrame.html, but contains our company logo, which we will place in the img directory. For this example, we generate a logo in JPG format with dimensions of 142 by 78. Follow these steps:

1. Place the logo file, called logo.gif, into the img directory.
2. Copy the borderFrame.html file in the allLang directory to logoFrame.html and place it in the allLang directory.
3. Edit logoFrame.html and change the lines:

```
<BODY BGCOLOR="#d4d0c8">
</BODY>
```

to

```
<BODY BGCOLOR="#d4d0c8" topmargin="0" leftmargin="0">
<p></p>
</BODY>
```

4. Open im.html in each language sub-directory, and change the top three lines from:

```
<HTML>
<frameset rows='8,*,18' frameborder=0 border=0 framespacing=0>
<frame src=../allLang/borderFrame.html noresize scrolling=no
target="_self">
```

to

```
<HTML>
<frameset rows='78,*,18' frameborder=0 border=0 framespacing=0>
<frame src=../allLang/logoFrame.html noresize scrolling=no target="_self">
```

If you are supporting multiple languages, you will need to perform these changes to the im.html in the other language directories you plan to use.

After making these changes, you may find that the IM Chat window is too small for the transcript text, so we also want to change the default launch size for the window. Do this by opening the file `res.js` and changing the line:

```
var WIN_IM_HEIGHT = 280;
```

to

```
var WIN_IM_HEIGHT = 350;
```

This results in the IM chat window shown in Figure 9-31.

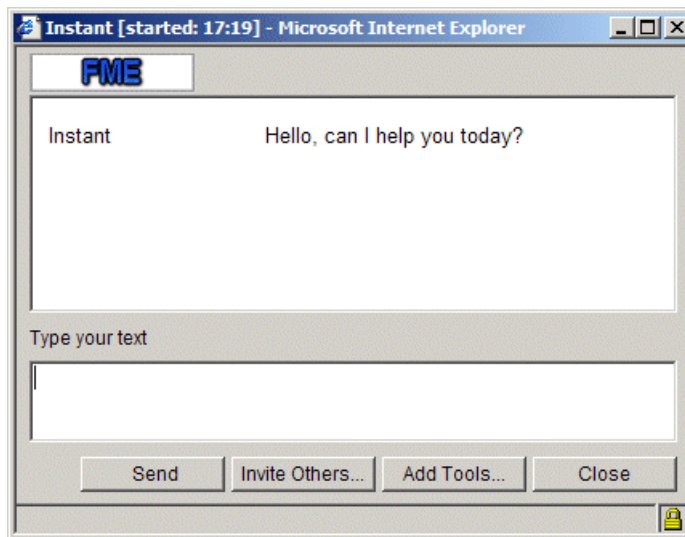


Figure 9-31 Modified instant message window

Remove the Invite Others... and Add Tools... buttons

In “chatBtn.html” on page 231, we can see that this HTML page generates the buttons for the chat window based upon the type of chat. To keep the Invite Others... and Add Tools... buttons from appearing, we will simply remark out the code for those buttons. Do the following steps:

1. Open the file `chatBtn.html` in each language sub-directory and change the following lines from:

```
if (!top.isAnonymous || top.anonCanResolve)
    document.writeln('<td align=right><input type=button .... Truncated...
if (top.meetingsEnabled)
    document.writeln('<td align=right><input type=button .... Truncated...
to:
//if (!top.isAnonymous || top.anonCanResolve)
```



```
// document.writeln('<td align=right><input type=button .... Truncated...
//if (top.meetingsEnabled)
// document.writeln('<td align=right><input type=button .... Truncated...
```

This will now keep the Invite Others... and Add Tools... buttons from appearing in the customer's chat window, as shown in Figure 9-32.

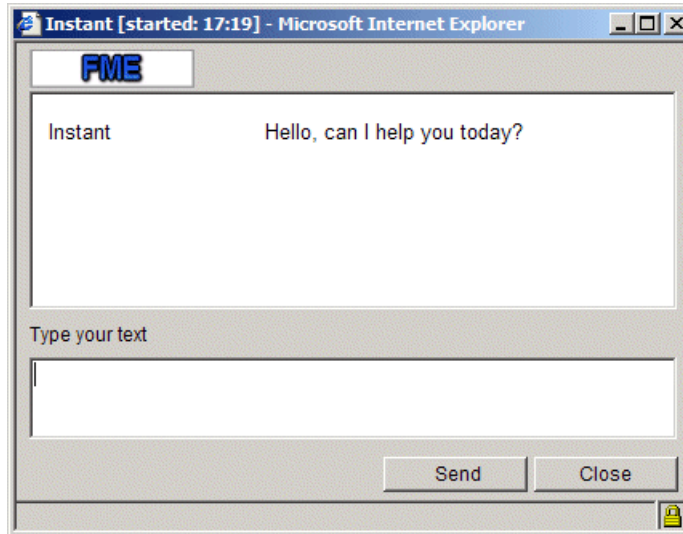


Figure 9-32 Instant message with no invite or tools buttons

Allow an agent to see who is on their Web site

Now that we have customized the experience for the customer, we need some way for the agents to see who is online and if they so wish to initiate a conversation with them. To keep track of who is on the Web site we will make use of Sametime's ability to accept anonymous logins and its ability to track Places. The easiest way to think of Places are as physical rooms, where when you can enter a room and see how many people are in there and who they are. One thing to keep in mind if you do think of Places this way is that with Sametime Places, people do have the ability to be in more than one Place at the same time. To keep this sample simple, we are going to have one Place that we are going to call "WEBSITE". When a person visits a Web page, this is the Place where they will enter, and this is the Place the agent will watch to see who is on the Web site.

To complete this task, there are a few steps we need to complete:

1. Modify the Web site pages to load the Sametime Links applet.
2. Log the visitor into Sametime.
3. Upon a successful login, move the customer into the Place "WEBSITE".

4. Provide a Web page for the agent to log in and watch visitors.
5. Display, on the agents page, the number of people in the "WEBSITE" Place.
6. Modify the Web site pages to load the Sametime Links applet.

We need to log the customer into Sametime via Sametime Links to track each page they visit on our Web site, so within the <HEAD> section of the Web page we need the following:

```
<LINK REL=STYLESHEET HREF="<CODEBASE>/stlinks.css" TYPE="text/css">
<SCRIPT src="<CODEBASE>/stlinks.js">
```

Where CODEBASE is the URL of the directory where the ST Links run-time files are:

```
http://<STSERVERNAME>/sametime/customerstlinks
```

This code provides the stylesheet and the JavaScript library for the Sametime Links functions.

Adding the function setSTLinksURL is a key part of ensuring the correct HTML pages are loaded, because all the Sametime Links chat windows and other dialogs are all HTML pages. The function setSTLinksURL tells the Sametime Links where these HTML files are located on the Sametime server:

```
<SCRIPT>
setSTLinksURL("<ORIGINALCODEBASE>", "EN", "<CODEBASE>");
</SCRIPT>
```

Where ORIGINALCODEBASE is the URL of the directory where the ST Links run-time files are:

```
http://<STSERVERNAME>/sametime/stlinks
```

CODEBASE is the URL of the directory where the customized customer ST Links files are:

```
http://<STSERVERNAME>/sametime/customerstlinks
```

For complete details on the stSTLinksURL function and its parameters, please refer to the stSTLinksURL function description within the *Sametime Links Javascript API Reference* included in Sametime Software Development Kit (SDK), which is installed on your Sametime server.

Note: For information about how to access this document, please refer to 2.2.1, "Sametime Software Development Kit (SDK) documentation" on page 22. This document can also be access via the following URL on your Sametime Development Server, assuming that the SDK has been installed:

```
http://<your sametimeserver name>/sametime/toolkits/st31linkstk/index.html
```

Log the visitor into Sametime

To log the user into Sametime, we need to use the `writeSTLinksApplet` function, which is documented in the *Sametime Links 3.1 Toolkit Developer's Guide and Reference*. For our sample, customers are going to be logged in as anonymous users, so we need to add the following into the HTML of our page:

```
<SCRIPT>
writeSTLinksApplet ("", "", false);
</SCRIPT>
```

What this does is log the customer in as anonymous, where their names take the form of `Userxxx/Guest`, where `xxx` is an number that increments by one for each visitor. There are ways to set the name for the visitor, but we will not cover that here; refer to the *Sametime Links 3.1 Toolkit Developer's Guide and Reference* for more details.

Move the customer into the Place "WEBSITE"

If the user is logged in successfully to Sametime, Sametime Links fires the function `STLinksLoggedIn`; by waiting for this function to be called before trying to enter the Place "WEBSITE", we can ensure that the Sametime Links applet has loaded successfully and avoid any JavaScript errors that may occur otherwise. With the following code within the HTML page, we can log the customer into the "WEBSITE" Place:

```
<SCRIPT>
function STLinksLoggedIn()
{
    STLinksEnterPlace("WEBSITE", true);
}
</SCRIPT>
```

The first parameter of this function is the name of the Place we want the customer to enter. The second parameter makes Sametime Links wait 30 seconds before taking the user out of the Place. This ensures that as the customer navigates between pages, Sametime Links has time to enter the Place again after the new page loads. This way, when the customer navigates between pages that enter the same virtual Place, the customer will remain in the Place during navigation.

Provide a Web page for the agent to log in and watch visitors

Now that we have the customer entering the Place, we need some means for the agents to see how many people are in the "WEBSITE" Place and who they are. To do that, we have to perform similar steps to setting up the customers page.

First, we need to log the agent into Sametime via Sametime Links on a custom page for monitoring the Web site. So we first need to create a standard HTML file

that the agent will load. Within the <HEAD> section of this page, we need the following:

```
<LINK REL=STYLESHEET HREF="<CODEBASE>/stlinks.css" TYPE="text/css">
<SCRIPT src="<CODEBASE>/stlinks.js">
```

Where codebase is the URL of the directory where the ST Links run-time files are:

```
http://<STSERVERNAME>/sametime/agentstlinks
```

As with the customer pages, we need to add the function setSTLinksURL to ensure the correct HTML pages are loaded. The function setSTLinksURL tells the Sametime Links where these HTML files are located on the Sametime server:

```
<SCRIPT>
setSTLinksURL("<ORIGINALCODEBASE>","EN",<CODEBASE>");
</SCRIPT>
```

Where ORIGINALCODEBASE is the URL of the directory where the original ST Links run-time files are:

```
http://<STSERVERNAME>/sametime/stlinks
```

CODEBASE is the URL of the directory where the customized agent ST Links files are:

```
http://<STSERVERNAME>/sametime/agent st links
```

For complete details on stSTLinksURL function and its parameters, please refer to the stSTLinksURL function description within the *Sametime Links Javascript API Reference*, which is included in Sametime Software Development Kit (SDK) (see 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22 for more details).

Next, we need some means of getting the agent's Sametime login details. For the purpose of this sample, the easiest way is to use JavaScript's **prompt** command. So we need some variables to store the agent's user name and password, the prompt functions to get them from the users, and the writeSTLinksApplet to log the agent in. The code in Example 9-4 demonstrates how to do these tasks.

Example 9-4 Prompting for login details

```
<script language="JavaScript1.2">

    //Declare variables that will be used to store agent details
    var STUserName, STPassword;

    //Prompt for Agents name
```

```
STUserName = prompt("Enter Sametime Login Name.", "");

//Prompt for Agents password
STPassword = prompt("Enter Sametime Password.", "");

//Using the variables storing the name and password login to STLINKS
writeSTLinksApplet(STUserName, STPassword, false);

</script>
```

Display the number of people in the "WEBSITE" Place

We need a Place on the page to let the agent see how many people are in the Place "WEBSITE", which is done using the Sametime Links function `writePlaceCounter` within the page. The following HTML on the agents page will allow you to see how many people are in the Place "WEBSITE":

```
<script>
writePlaceCounter("WEBSITE", "WEBSITE", false);
</script>
```

The first parameter being passed to the `writePlaceCounter` is the name identifying the Place, and the second parameter is the display name for the Place, which appears in the window title when the Place counter is clicked on. The third parameter is if you want to keep the person in the Place for 30 seconds before being seen as having left the Place; as this is an agent, we will set it to `false`. By using the function `writePlaceCounter` within a page, it automatically logs you into the Place to get the status. So if an agent loads this page, and there are no other agents or customers logged on to the site, the Place counter will indicate "1", which indicates they are in the Place. If another agent, or a customer enters the Place, the counter will increase by 1 to "2".

By clicking on the counter, the window shown in "place.html" on page 241 appears, displaying who the people are in the "WEBSITE" Place. For this simple example, it is easy tell agents from customers, as all the customers will have names like `Userxxx/Guest`.

In the `place.html` dialog, the agent can click on a customer's name to initiate an instant messaging conversation with them. From that conversation, it is possible for the agent to start screen sharing or send the customer URLs, or even involve other people in the conversation through the use of n-way chat. You can even start a meeting using audio/video and screen sharing to demonstrate how to do something.

Now, we could stop right here, and from the customer's point of view you would have a very powerful, proactive interaction offering on your Web site, and if you were a single agent watching the Web site, then you will probably find this

perfectly suitable. However, in the real world, there tends to be more than one agent watching the site, which leads to internal issues for the application, which makes it awkward for real life use. Such issues include:

- ▶ No way to know if another agent is already chatting with a customer
- ▶ No way to know which page the customer is on
- ▶ No information concerning the customer or where they came from

Allow the agent to see which page the customer is on

There a number of ways this can be done. The simplest is to add a line to the customers HTML pages, which enters them into a Place with the name of the Web page so that our existing HTML:

```
<SCRIPT>
function STLinksLoggedIn()
{
    STLinksEnterPlace("WEBSITE",true);
}
</SCRIPT>
```

would change to:

```
<SCRIPT>
function STLinksLoggedIn()
{
    STLinksEnterPlace("WEBSITE",true);
    STLinksEnterPlace("name of webpage here",true);
}
</SCRIPT>
```

Then, on the agent's HTML page, we build a list of writePlaceCounters on a HTML page for each Web page you have in your site, then, with that single page, the agent would be able to watch the number of people on the different pages, and then click on the page to see who is in there.

We also have another option that does not involve creating any more Places, and makes use of Sametime's ability to add a status message to someone's status.

By default, visitors to your site that are logged in with Sametime Links have a status message of userxxx/Guest is active by adding a single line of script to each Web page, we can change the customer's status to have whatever information we want, such as the page the person is on or if they are chatting with an agent already. For this example, we will start by updating the person's status whenever they load a page, so that agent can see which page they are on by simply hovering over their name in the place.html dialog.

Currently, our existing customer Web page HTML contains:

```

<SCRIPT>
function STLinksLoggedIn()
{
    STLinksEnterPlace("WEBSITE",true);
}
</SCRIPT>

```

We change this to:

```

<SCRIPT>
function STLinksLoggedIn()
{
    STLinksSetMyStatus("32","On Page: " + window.location.pathname);
    STLinksEnterPlace("WEBSITE",true);
}
</SCRIPT>

```

There are two parameters passed to the function STLinksSetMyStatus, the first of which is a numeric constant for representing the users status. These constants are:

```

0 - Offline
32 - Active
64 - Not using the computer
96 - Away
128 - Do not disturb (DND)
544 - Mobile active
608 - Mobile away

```

The second parameter is the status message, which is where you normally see messages such as "I am in a meeting."

By changing the user's status when they enter a page, and hovering over the customers name in the people list, the agent can now see the whole path name of the page the customer is on, which tends to be more than what most people want to see, so with a couple of additions, we can return the last section of the path details:

```

<SCRIPT>
function STLinksLoggedIn()
{
    var currentPath = window.location.pathname;
    var currentPage = currentPath.substring(currentPath.lastIndexOf('/') + 1);
    STLinksSetMyStatus("32","On Page: " + currentPage);
    STLinksEnterPlace("WEBSITE",true);
}
</SCRIPT>

```

If you understand JavaScript, you will realize you can start to use this in combination with cookies and other variables to add other valuable information, such as:

- ▶ The time the customer first came to the site
- ▶ The time the customer went to their current page
- ▶ The Web site that referred them to your site
- ▶ The customers host name, for example, www.ibm.com
- ▶ The customers IP address, for example, 219.243.221.67

Note: The STLinksSetMyStatus is referred incorrectly in the Sametime Links documentation and Sametime Links release notes as STLinksChangeMyStatus, which is the same function, just documented with the wrong name. The function name you need to use is STLinksSetMyStatus.

Let agents know when an agent is chatting with a customer

Currently, when an agent is chatting with a customer, there is no way for other agents to know this. Again, making use of the status information available within Lotus Sametime, we will make it easy for other agents to see when a customer is already chatting with an agent. As you could see earlier, there are a number of different states a Sametime Links user status can have. As our customers are not going to be running Sametime Links over mobile devices, we'll make use of the 544 status, which is the Mobile Active state:

544 - Mobile active

To change the status when a person is in conversation with an agent, we need to grab the point where the agent starts chatting with the customer. We can intercept this through the stlinks.js file that is in the /customerstlinks directory. Within that JavaScript library, there is a function called openImWindow. This function is called when someone sends a message to someone else. In the case of our Web site visitors, this can only be an agent sending them a message, so we know that when this function is called that an agent has started a conversation with a customer. To let other agents know this, we will change the customers state to Mobile Active, and indicate in their status message which agent they are chatting with. We do this by changing the function openImWindow in the customer stlinks.js file from:

```
function openImWindow (partnerId, partnerName, away, mobile, isSecured,
winIndex, winName)
{
    partnerName=escape(partnerName);
    PartnerId=escape(partnerId);
    Open(urlPages+"/chatWindow.html?winType=IM&urlMain="+urlMain+"....
Truncated
....
....
```



```

....
.... Left="+winIndex*20);
}

to:

function openImWindow (partnerId, partnerName, away, mobile, isSecured,
winIndex, winName)
{
    partnerName=escape(partnerName);
    PartnerId=escape(partnerId);
    Open(urlPages+"/chatWindow.html?winType=IM&urlMain="+urlMain+"....
Truncated
....
....
....
.... Left="+winIndex*20);
    STLinksSetMyStatus('544','Chatting with: ' + unescape(partnerName));
}

```

This will now display the mobile status icon and status message *Chatting with: Agents Name* when a customer is engaged in conversation.

9.5.2 Taking it further

The great thing about Sametime Links is how most of the code and the dialog appearance is stored within the STLINKS directory, and totally customizable. This chapter has gone so far, but do not stop there. Using this as the basis for working with Sametime Links, it is possible to take the core Sametime Links package and extend its functionality a great deal, as can be seen in tools such as the WebSphere Portal Collaboration Center. Some IBM Lotus Partners have also extended Sametime Links to include features such as:

- ▶ Rich text and emoticon support
- ▶ The ability to chat with a customer, and not have the customer see the internal person's name
- ▶ Alerts for when people enter specific pages on the Web site, such as support or purchase
- ▶ The ability for a customer to request help, rather than having to wait until it is offered
- ▶ The ability for the agent to send URLs to the customer that then automatically open in a new Web browser
- ▶ The ability to log customer visits and chats into a database

9.6 Using Sametime Links with bots

One request many people have when using Sametime Bots is the ability to access them via a Web browser. Sametime Links is obviously a great way of providing access to Sametime and Sametime Bots on a Web page, but Sametime Links works differently for traditional Sametime Connect clients when it initiates a conversation, making bots harder to interact with. In this section, we will discuss how Sametime Links can be modified to work with Sametime Bots.

9.6.1 How does it differ?

So how is a Sametime Links conversation different from a conversation initiated by the Sametime Connect client? With a traditional conversation that is initiated by the Sametime Connect client, the first thing that happens when the Connect client initiates a conversation with another Sametime user or bot is that the Sametime Connect client opens the IM session. What this means is that the receiving party receives an `imOpened` and `imReceived` event on the `imListener`, which can then kick off other processes (in the case of bots, often displaying a welcome message). As you can in the 3.3.1, “The Echo Bot” on page 56 example, the `imReceived` event is used to send back a welcome message. Where the Sametime Links client differs is that the Sametime Links client does not kick off the `imOpened` or `imReceived` event until an actual message is sent. This obviously leads to some issues for bot developers who are expecting the first `imOpened` and `imReceived` events to just be the opening of the IM conversation rather than being the actual query itself.

The easiest way around this problem is to modify the Sametime Links HTML files slightly so that when the chat window is opened it immediately sends a text message to the bot. In Figure 9-33 on page 257, we can see this `chatWindow.html` has been modified to send the word “HELP” when the bot link is clicked, thus allowing the bot to send its welcome message and initiate the conversation with the Sametime Links user. We can make this change by modifying the `chatWindow.html` file.

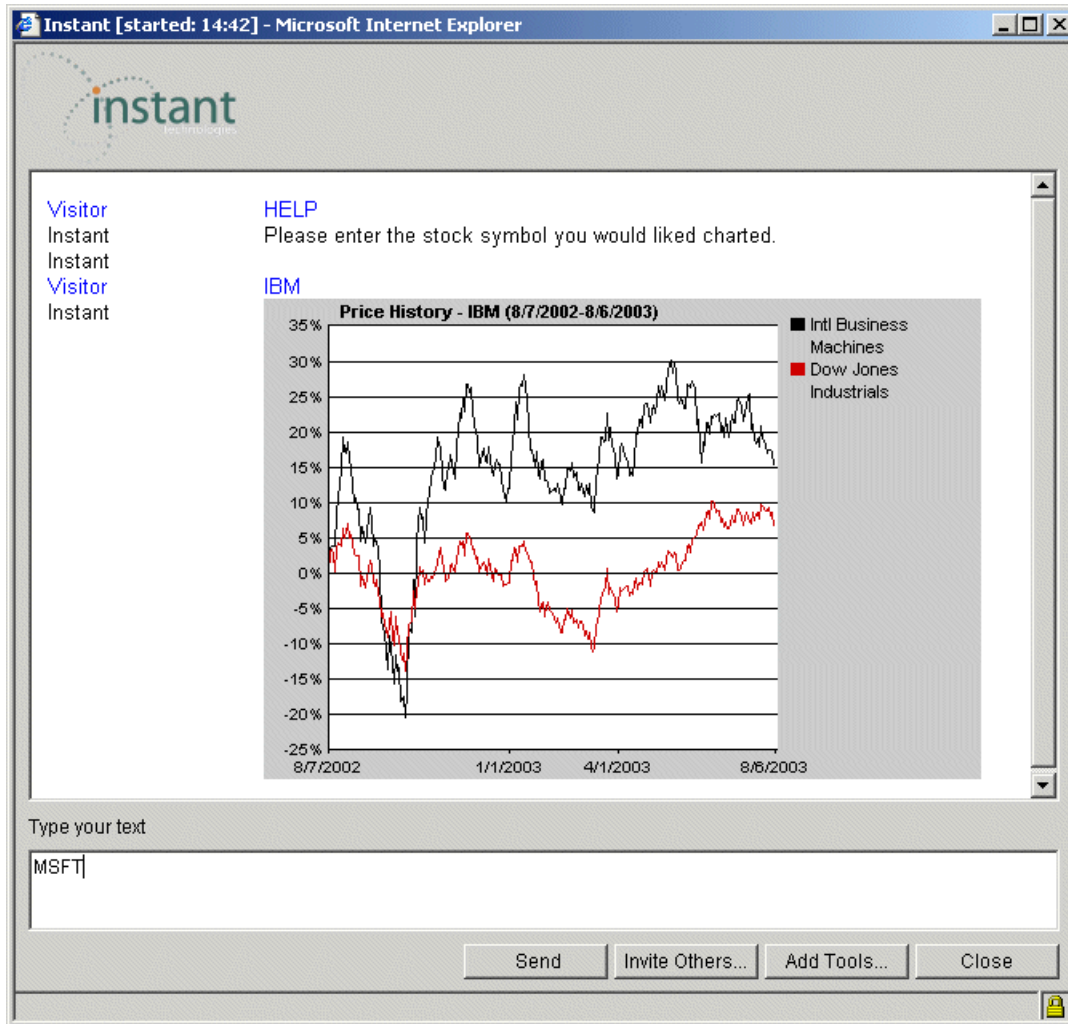


Figure 9-33 Instant Technologies Stock Bot running in Sametime Links

9.6.2 Changing the chatWindow.html

Before we make changes to the chatWindow.html file, we need to think about what impact this will have, and if we want this effect to happen every time we use Sametime Links or only when we are using bots. If we were to make these changes to the default Sametime Links HTML files in the STLINKS directory, then every time you clicked on a name with Sametime Links, the person you were starting a conversation with would receive a message even before you had typed anything. This obviously would appear very strange to the person receiving

the message and would be inconsistent with the way the other Sametime Connect worked. For this reason, we recommend that you create a new copy of the Sametime Links directory *serverdatadirectory/domino/html/sametime/stlinks*; for this example, we would recommend naming the new directory *serverdatadirectory/domino/html/sametime/bots* so it is easy to remember why it exists. By copying this directory, we will be able to refer to this directory for HTML pages when we use Sametime Links to communicate with bots. To reference this page rather than the other stlinks directory, we will need to change the line that calls the setSTLinksURL function. The function setSTLinksURL tells Sametime Links where the HTML files used by Sametime Links are located. Using the directory we specified above, we would change the default of:

```
<SCRIPT>
setSTLinksURL("sametimeserver/sametime/stlinks");
</SCRIPT>
```

to:

```
<SCRIPT>
setSTLinksURL("sametimeserver/sametime/bots");
</SCRIPT>
```

By doing this, we are letting the Sametime Links applets and functions know that all the HTML files that Sametime Links requires are within that directory.

Now that we have pointed the setSTLinksURL to the right location, we need to modify the HTML file that will send the initial message to the bot. The easiest way to have the page send a message to the bot (when the bot is clicked) is to edit the HTML file chatWindow.html. This file exists within each language subdirectory of the new Sametime Links directory we created (bots), and we will need to edit the chatWindow.html file in any language subdirectory we intend to use. For the purposes of this redbook, we will just go through editing the English version of the chatWindow.html. To edit this file, first find it in the en subdirectory in the bots directory you created earlier. Once you have located the file, open it with a text editor such as Notepad. Upon opening the file, look for the lines in Example 9-5.

Example 9-5 stlinksAppletIsUp function

```
function stlinksAppletIsUp()
{
    stlinksAppIsUp = true;
    chatEnabled = true;
    if (inputAppIsUp)
        getInputApplet().setEnabled(true);
}
```

The `stlinksAppletIsUp` function in Example 9-5 on page 258 are called when the Sametime Links applet is loaded and active within the chat window. If we were to try and use the Sametime Links applet before it was loaded, you would receive a Java error. To avoid that error, we want to place our code to send a message to the bot within this function. To do this, we literally have to add one line:

```
sendText('');
```

We will add this line right before the end of the function, as seen in Example 9-6.

Example 9-6 Modified `stlinksAppletIsUp` function

```
function stlinksAppletIsUp()
{
    stlinksAppIsUp = true;
    chatEnabled = true;
    if (inputAppIsUp)
        getInputApplet().setEnabled(true);
    sendText('');
}
```

This function sends an empty string via the `sendText` function to the Sametime Bot. In a real world situation, you would modify your bot code to handle receiving an empty string from Sametime Links clients so that it does not respond with search results the first time it is called.

9.7 Adding menu options to Sametime Links

When people deploy Sametime Links, they often find they want to add extra features to the live names they are displaying on a page. Since Sametime Links is built around a collection of JavaScript and HTML, it is very flexible for allowing this kind of customization. In this section, we will give an example of customizing the live names to allow you to send e-mail to an offline person, share an application, and add the name of the person to your contact list. Overriding the default Sametime Links click behavior, we will display a pop-up menu using dynamic HTML and the Sametime Links JavaScript API.

Note: The content and samples for this section were adapted from “Adding a Popup Menu to your Sametime Links” by Haim Schneider in *LDD Today*, July 2003. To reference the original article, see the following URL:

http://www.lotus.com/1dd/today.nsf/lookup/ST_popup

To access the The Lotus Developer Domain (LDD), visit:

<http://www.lotus.com/1dd>

9.7.1 The sample pop-up menu

The sample pop-up menu that we are going to add has the following four menu items:

- ▶ Message
Sends an instant message to the selected person.
- ▶ Share Application
Invites the selected person to an application-sharing meeting.
- ▶ Send Mail
Creates a new e-mail message for the selected person with the user's default e-mail client.
- ▶ Add to Contact List
Adds the selected person to the user's Sametime Connect's contact list.

Figure 9-34 shows the sample pop-up menu.

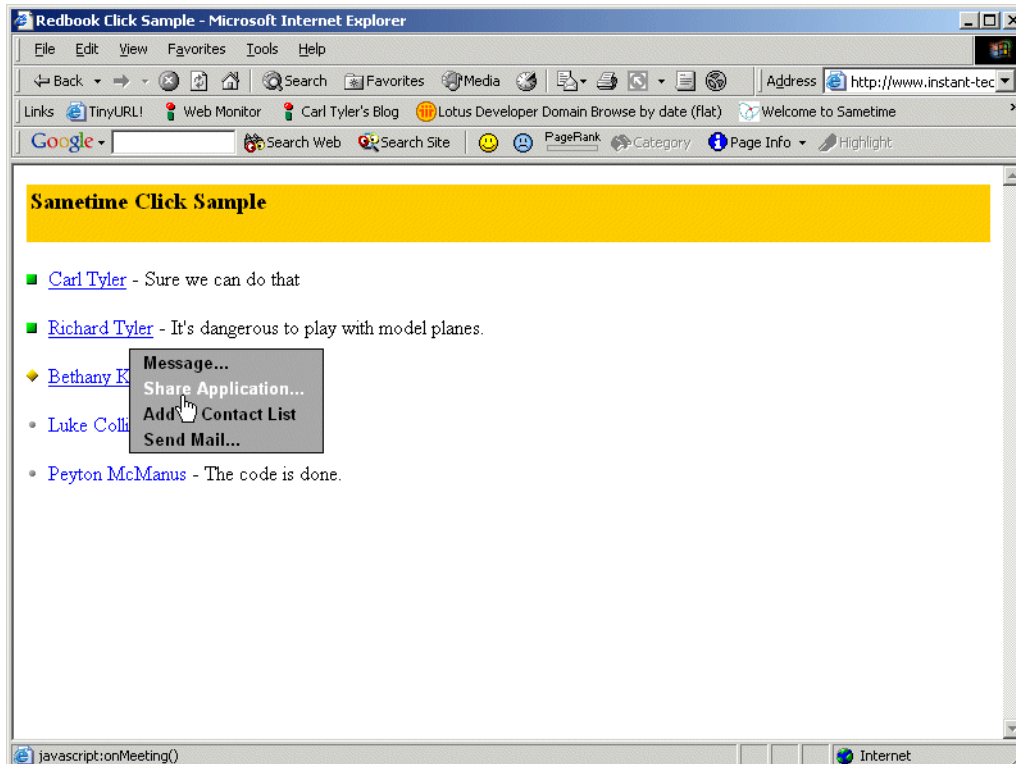


Figure 9-34 Customized Sametime Links click menu

When a person is offline, Sametime Links are normally displayed as simple HTML text. They are not clickable links, because you cannot send an instant message to an offline person. After implementing the pop-up menu, however, this can be changed. We want the menu enabled for offline names, because some of the options are applicable to offline people. For example, you can send e-mail to an offline person or add that person to your contact list. Accordingly, we will change the default behavior of the links and make offline names display as HTML clickable links. When an offline name is clicked, the menu items that are only applicable to online people will be disabled, as you can see in Figure 9-35.

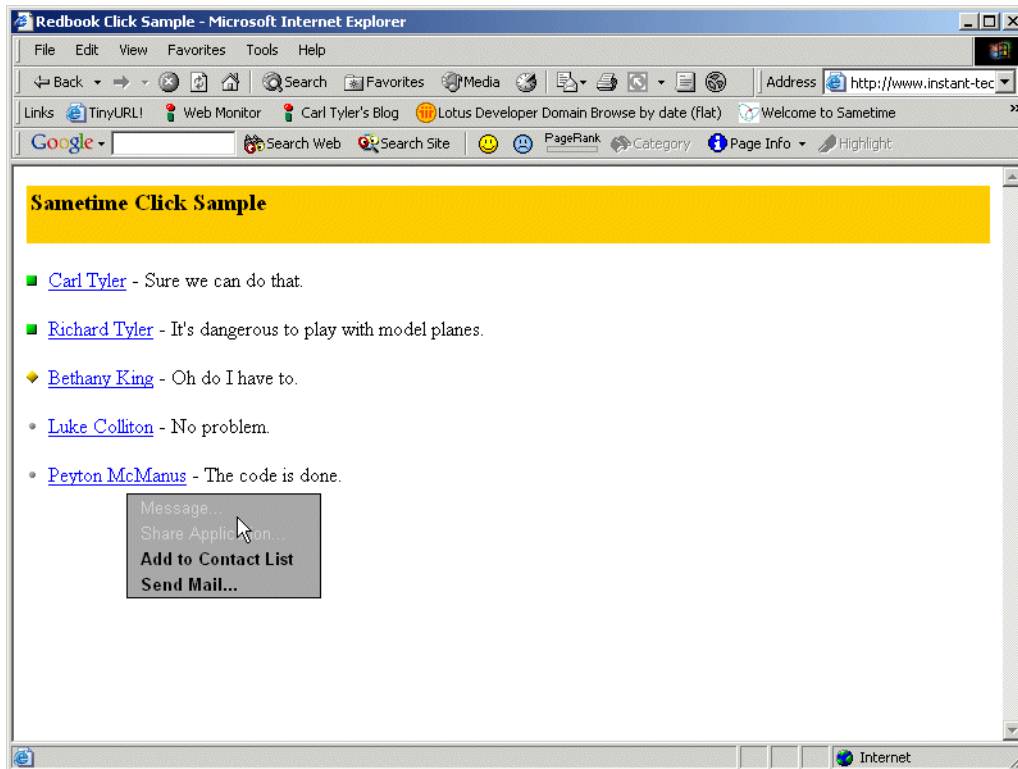


Figure 9-35 Offline HTML click menu

This example is easily customizable, and this section shows you how to implement the menu framework and how to add menu items. You can use this menu *as is* or add or replace items with others that you want to present in your application. You can add application-specific options, for example, showing a list of documents authored by the selected person or displaying a user profile.

9.7.2 Starting with a Sametime Links-enabled page

In this example, we start with a Web page that is already enabled with Sametime Links and add the pop-up menu to it. If your application is not yet enabled with Sametime Links, see the instructions in the *Sametime Links 3.1 Toolkit Developer's Guide and Reference*. Basically, you add several simple lines of HTML/JavaScript code to the Web page and provide the user's login name and password or single sign-on token. You can also start by adding the pop-up menu to one of the sample pages provided with the toolkit.

To access the *Sametime Links 3.1 Toolkit Developer's Guide and Reference* and the samples, go to the Sametime Links Toolkit home page. You can go to the Toolkit home page by selecting the **SDK** link in the Sametime server home page. In the Toolkits page, select the **Sametime Links Toolkit** link. If there is no link, then you have to install the Sametime Toolkits package on the Sametime server. You can also find the *Sametime Links 3.1 Toolkit Developer's Guide and Reference* and the toolkit download on the Toolkits and Drivers page.

Note: The Sametime Links run-time files are pre-installed on the Sametime 3.0 and 3.1 servers, so you only need to install the toolkit for the samples and the *Sametime Links 3.1 Toolkit Developer's Guide and Reference*. If you are using a Sametime 2.5 server, you have to install Sametime Links for Sametime 2.5.

Making offline names clickable links

By default, offline Sametime Links are displayed as normal HTML text. To provide a menu for offline Sametime Links, we need to make offline names clickable. We do this by adding the `offlineLink` option to the `writeSametimeLink` call:

```
writeSametimeLink("Haim Schneider/Haifa/IBM", "Haim Schneider", true,  
"offlineLink:yes")
```

This option should be used only if you want to override the click behavior (which we are going to do in the next step). Otherwise, when the link is clicked, Sametime Links would start an instant message with the offline user, which would, of course, fail.

With the `offlineLink` option, offline names are displayed as normal HTML links, namely, blue underlined text. There are now two styles of links: bold green links for online people and normal blue links for offline people. The mixed style may be confusing, so we are going to change the style of online links and make it the same style used for offline links. Although the green bold style is normally used to highlight online names, the status icon next to the name is by itself a good indication of the online status.

The styles for online and offline links are defined in the cascading style sheets file `stlinks.css`. You can modify the online style in this file, and this will change the styles on every Web page that links to this style sheet. Alternatively, you can override the style in your page by adding the style definition in Example 9-7.

Example 9-7 Online and offline styles

```
<STYLE>
a.online:link {color:blue; font-weight:normal}
a.online:visited {color:blue; font-weight:normal}
a.online:hover {color:blue; font-weight:normal}
a.offline {color:blue; font-weight:normal}
</STYLE>
```

Put the style definition in Example 9-7 after the `<LINK>` tag that links to `stlinks.css` so that it overrides the online style definition in that file.

Handling the click event

We use the event `STLinkClicked` to override the default click behavior. To handle one of the event types of the Sametime Links Toolkit, you add a function with the name of the event to your page. The function should have the parameters specified by the event type. To handle the click event, we add an implementation of `STLinkClicked` (Example 9-8 on page 264). This function is called whenever a Sametime link is clicked. Note that unlike all other events in the Toolkit, `STLinkClicked` overrides the default behavior. Instead of starting an instant message, which is the default behavior, our function is called. For all other events, custom event handlers are called after the default handler is called.

Table 9-2 STLinkClicked event parameters

Parameter	Description
userName	The unique user name as provided in the <code>writeSametimeLink</code> call
displayName	The display name of the user
status	A numeric constant indicating the current status of the user (See the list of status values in the Developer's Guide.)
evt	The JavaScript Event object that provides information on the click event, such as the mouse position

We use a global variable called `selectedName` that stores the name of the user who was clicked. This variable is used by the function that draws the menu and the functions that implement the menu items actions.

Example 9-8 STLinkClicked function

```
var selectedName = "";

function STLinkClicked(userName, displayName, status, evt)
{
    selectedName = userName;
    showMenu(userName, displayName, status, evt.clientX, evt.clientY );
    evt.cancelBubble = true;
}
```

Our implementation of the STLinkClicked event simply sets the selectedName variable and calls showMenu to display the menu. We also have to set the event object's cancelBubble property to true in order to cancel bubbling of the event beyond the link itself.

Displaying the menu

Before adding the menu HTML code, we add style definitions for the menu box, a menu item, and a disabled menu item (Example 9-9). Add the styles at the head section of the page.

Example 9-9 Menu box style definition

```
<STYLE>
.stMenu {background-color:darkgray; position:absolute; visibility:hidden;
border-Width:1px; border-Style:solid; border-Color:black}
a.menuItem {color: black; font: 10pt Arial; font-weight:bold;
text-decoration:none;}
a.menuItem:hover {color: white}
.menuDisabled {color: lightgrey; font: 10pt Arial; font-weight:normal;
text-decoration:none;}
</STYLE>
```

By using the style definition in Example 9-7 on page 263 we made the menu background dark gray with black bold text for the menu items. The text of a highlighted menu is displayed in white, and disabled menu items are displayed as light gray plain text. The colors, fonts, and other style attributes can be easily modified, so if you prefer your menus green on a purple background with a yellow border, just change the style definitions.

Note: By default, HTML links are underlined; to make the text look more like a menu item, we have to remove the underline. We do this by setting the text-decoration attribute to none.

Now that we have the styles defined, we can add the menu. The menu is a <DIV> tag that contains a table, and each menu item is a table cell. The <DIV>

contents are shown in Example 9-10. You can place the menu <DIV> tag anywhere in your page. Its initially hidden because it uses the stMenu style, which has a visibility: hidden attribute.

Example 9-10 Menu table definition

```
<DIV id=menu class=stMenu>
<TABLE border=0 cellPadding=2 cellSpacing=0 width=150>
<TR><TD id="menu_message"></TD></TR>
<TR><TD id="menu_meeting"></TD></TR>
<TR><TD id="menu_addContact"></TD></TR>
<TR><TD id="menu_mail"></TD></TR>
</TABLE>
</DIV>
```

The menu is initially empty. As you can see in Example 9-10, there are four table cells for the four menu items, but these cells are empty. The cells are dynamically populated when the menu is opened. This allows us to enable or disable the menu items according to the online status and to create the Send Mail menu item with the e-mail address of the selected person.

The setMenuItem function in Example 9-11 sets the content of a single menu item.

Example 9-11 setMenuItem function

```
function setMenuItem(itemId, text, action, isEnabled)
{
var menuItem = document.getElementById(itemId);
//add spaces to the left of the text
text = "&nbsp;   " + text;
if (isEnabled)
menuItem.innerHTML = "<A class=menuItem + href='" + action + "'"> + text +
"</A>";
else menuItem.innerHTML = "<SPAN class=menuDisabled>" + text + "</SPAN>";
}
```

The ID of the table cell (the <TD> tag) is used to access the item. SetMenuItem populates the cell with the menu item content. An enabled item is displayed as an HTML link with the action specified in the action parameter. The action is either a JavaScript URL or a mailto: link. A disabled item is displayed as plain text, using the menuDisabled style, which we defined in Example 9-11 on page 265.

Now let us look at the showMenu function in Example 9-12.

Example 9-12 showMenu function

```

function showMenu(userName, displayName, status, x, y )
{
    var isOnline = (status != 0);

    setMenuItem("menu_message", "Message...", "javascript:onMessage()",
isOnline);
    setMenuItem("menu_meeting", "Share Application...",
"javascript:onMeeting()", isOnline);
    setMenuItem("menu_addContact", "Add to Contact List",
"javascript:onAddContact()", isLoggedIn);
    setMenuItem("menu_mail", "Send Mail...", "mailto:" +
getEmailAddress(userName), true);

    var theMenu = document.getElementById("menu");
    theMenu.style.left=x - 10 + document.body.scrollLeft;
    theMenu.style.top=y + 10 + document.body.scrollTop;
    theMenu.style.visibility='visible';
}

```

Before displaying the menu, showMenu sets the content of the four menu items. It positions the menu at the point on the screen where the mouse was clicked, and finally makes the menu visible.

The first two items, Message and Share Application, are disabled if the status parameter is zero, which means that the selected person is offline. The Add to Contact List item is disabled if the page user is not logged in. The user has to be logged in with the Sametime Links applet in order to add a person to the contact list. In showMenu, we use the flag isLoggedIn to test whether or not the user is currently logged in. This flag is not provided by the API. Because there is no direct API call for testing if the user is logged in, we use the STLinksLoggedIn and STLinksLoggedOut API events to set the isLoggedIn flag, as can be seen in Example 9-13 on page 267.

Example 9-13 Providing logged in status

```
var isLoggedIn = false;

function STLinksLoggedIn(myUserId, myUserName)
{
    isLoggedIn = true;
}

function STLinksLoggedOut(reason)
{
    isLoggedIn = false;
}
```

Menu options implementation

Next, we implement the functions that are used as the actions for our sample menu items, `onMessage`, `onMeeting`, and `onAddContact`. As you can see in Example 9-14, these functions are very easy to implement with the Sametime Links Toolkit.

Example 9-14 onMessage, onMeeting, and AddContact functions

```
function onMessage()
{
    STLinksCreateIM(selectedName);
}

function onMeeting()
{
    STLinksCreateMeeting(selectedName, "chat;share:whiteboard", "Meeting",
        "Please join the meeting");
}

function onAddContact()
{
    STLinksAddToContactList(selectedName, "Work" );
}
```

The function `onMeeting` creates an application-sharing meeting with the selected person. The parameters passed to `STLinksCreateMeeting` are the names of the invitees, the list of meeting tools (we selected chat, application sharing, and whiteboard), the meeting topic, and the invitation message.

The function `onAddContact` adds the selected person to the user's contact list. The name is added to the "Work" group. If the user does not have a personal group named "Work", the group is automatically created. For simplicity sake, we used a hardcoded group name, but if you want, you can let the user choose

which group the user is added to. The Sametime Links Toolkit allows you to get the list of the user's personal groups. To get the list of groups, you call `STLinksGetPrivateGroups`. The result is returned in the `STLinksPrivateGroupsReceived` event. After you have the list of personal groups, you can change `onAddContact` so that it presents a dialog allowing the user to choose a group from the list of existing ones or enter a new personal group name.

Note: Getting the list of personal groups and adding a user to a personal group are the only things you can do with the contact list using the Sametime Links Toolkits. You will not find other functions in this API for retrieving the contact list or manipulating it, for example, adding public groups or removing users. These APIs were not added in order to keep the Sametime Links applet as small as possible. The Sametime Links Toolkit provides the main functionality needed for a client-side application. If you want to retrieve the contact list or manipulate it in your Web application, consider doing it on the server-side using the Sametime Community Server Toolkit. In a client application, you can use the Sametime Java Toolkit to access the contact list.

Getting the e-mail address

The Send Mail item is actually an HTML `mailto` link with the e-mail address of the selected person. Where does the address come from? Well, there is no magic way to get it. The e-mail address has to be placed on the page when the page is generated. We assume that the application that generates the page puts a mapping of names to e-mail addresses in the page. This mapping should include all the names that have Sametime Links in the page. We implement the mapping with a JavaScript object in Example 9-15. For example, if Carl Tyler and Bethany King are Sametime Links in the page, the application that generates the page adds JavaScript code that maps these names to their e-mail addresses.

Example 9-15 Defining the JavaScript object to hold addresses

```
<SCRIPT>
  var emailAddress = new Object();
  emailAddress["Carl Tyler"] = "ctyler@instant-tech.com";
  emailAddress["Bethany King"] = "bking@instant-tech.com";
</SCRIPT>
```

The names used in this mapping should be the same as the strings that are passed in the first parameter of the `writeSametimeLink` call. The function `getEmailAddress` in Example 9-16 on page 269 is used by `showMenu` to get the e-mail address of the selected person.

Example 9-16 getEmailAddress function

```
function getEmailAddress(name)
{
    return emailAddress[name];
}
```

Closing the menu

The only thing left to do now is to make sure that the menu is closed. Like any other menu, our menu needs to be closed when a menu item is selected or the mouse is clicked outside the menu area. In Example 9-17, we add a hideMenu function that hides the <DIV> tag when the menu needs to be closed.

Example 9-17 hideMenu function

```
function hideMenu()
{
    document.getElementById("menu").style.visibility='hidden';
}
```

How does hideMenu get called? The simplest way is to call it in the onClick attribute of the page's <BODY> tag:

```
<BODY onClick=hideMenu()>
```

Alternatively, if you do not want to change the <BODY> tag, you can set the onClick action with JavaScript by adding the code in Example 9-18 anywhere in the page.

Example 9-18 onClick action

```
<SCRIPT>
    document.onclick = hideMenu;
</SCRIPT>
```

How to use the sample page

Details on how to obtain this sample code are available in Appendix C, "Additional material" on page 467. The sample is based on one of the sample pages provided with the Sametime Links Toolkit. The original sample demonstrates how Sametime Links can be embedded in a message board page. For this section, we enhanced the page with our sample menu. Do not try to run the sample that way. It contains placeholders for the server address and people names that you have to complete before you can run it. The following steps will take you through the necessary process to make the sample work within your environment:

1. In the head section of the page, replace the three appearances of <your Sametime server> with the host name of the Sametime server. For example, if the host address of the Sametime server is sametime.acme.com, the head section of the page should contain what is shown in Example 9-19.

Example 9-19 Sametime server host name in head section

```
<LINK REL=STYLESHEET HREF=http://sametime.acme.com/sametime/stlinks/stlinks.css
TYPE="text/css">
<SCRIPT src="http://sametime.acme.com/sametime/stlinks/stlinks.js"></SCRIPT>
<SCRIPT>
    setSTLinksURL("http://sametime.acme.com/sametime/stlinks");
</SCRIPT>
```

2. In order for the links to show online users, change the Sametime link parameters Name1 through Name5 to the names of people who are registered in your directory. For example, replace:

```
writeSametimeLink("Name1", "Name1", true, "offlineLink:yes")
```

with:

```
writeSametimeLink("Carl Tyler", "Carl Tyler", true, "offlineLink:yes")
```

To ensure the names are unique, you can use the canonical name format. For example:

```
writeSametimeLink("Carl Tyler/US/Instant", "Carl Tyler", true,
"offlineLink:yes")
```

3. Fill in the e-mail address mapping. There are five mapping assignments for Name1 through Name5. Replace the names with the names that you used in the calls to writeSametimeLink. Replace the e-mail addresses with the corresponding e-mail addresses. For example, replace:

```
emailAddress["Name1"] = "email1";
```

with:

```
emailAddress["Carl Tyler"] = "ctyler@instant-tech.com";
```

4. Replace the login name and password in the writeSTLinksApplet call:

```
writeSTLinksApplet("<login name>", "<password>", false)
```

with the login name and password of a registered user. Note that although this sample uses password authentication, most Web applications enabled with Sametime Links use token authentication to achieve single sign-on and better security. See the *Sametime Links 3.1 Toolkit Developer's Guide and Reference* for instructions on how to use token authentication.

5. Put the page on a Web server and open it in a browser.

Note: You must use the URL, not the local file system address, to access the page. Sametime Links are not fully functional when you access an enabled page as a local file. When an enabled page is accessed as a local file, the Sametime Links are active, but a JavaScript error is displayed when you try to open a message window.

What we discovered by adding the pop-up menu

Adding a pop-up menu to Sametime Links allows you to add more functionality to live names in your application. This section walked you through the code you need to add to your page to implement the menu. In doing so, we covered several topics related to Sametime Links customization and the HTML/JavaScript API. Here is a list of the API topics that we covered:

- ▶ Displaying offline names as HTML links
- ▶ Changing the style used to display online names
- ▶ Overriding the click behavior
- ▶ Handling API events
- ▶ Using the API to add a user to the contact list, to create an instant message, and to create a meeting

The pop-up menu sample is a good example of the simplicity of the API. The sample goes beyond basic Sametime Links-enabling into what the Developer's Guide calls the advanced API, but as you can see, the advanced API is still easy to use. A single line of JavaScript code is all you need to do things like adding a person to the contact list, creating an instant message, or creating an application-sharing meeting.



Sametime-enabling portlets

This chapter covers the topic of integrating Sametime functionality within a portal and portlets. We describe the possibilities and approaches when using Sametime to enable instant collaboration within a portal framework. Using a basic portlet, we demonstrate two approaches to Sametime-enabling the names and menu links within the portlet. One approach is to use the IBM WebSphere Collaborative Components API, while the other approach leverages the Sametime Links API.

At a basic level, portlets can be considered as being similar to Web pages that can have instant collaboration capabilities. When developing with portlets, however, the developer has options and advantages that are not available for regular Web pages. We will cover these advantages in this chapter, while also highlighting advantages of each API and ultimately helping you to select the one that best meets your business requirements.

The following subjects are covered with this chapter:

- ▶ IBM WebSphere Portal Server overview
- ▶ IBM WebSphere Portal Server and Sametime
- ▶ Sametime-enabling portlets using the IBM WebSphere Portal Collaborative Components.
- ▶ Sametime-enabling portlets using the STLinks API

10.1 IBM WebSphere Portal Server overview

This section provides a basic overview of IBM WebSphere Portal and portlets in order to establish a foundation for understanding the capabilities and issues with integrating Sametime functionality into portlets. Please note, however, that the objective of this chapter is not to provide an in depth review of IBM WebSphere Portal. Please refer to the following sources for more in depth coverage:

- ▶ *IBM WebSphere Portal V4 Developer's Handbook*, SG24-6897
- ▶ *IBM WebSphere Portal Version 4.1 Handbook Volume 1*, SG24-6883
- ▶ Info Center for IBM WebSphere Portal Version 4.2.1, found at:
<http://publib.boulder.ibm.com/pvc/wp/42/exp/en/InfoCenter/index.html>

10.1.1 IBM WebSphere Portal overview

IBM WebSphere Portal allows you to establish customized portals for your employees, Business Partners, and customers. As illustrated in Figure 10-1, the framework architecture implemented in this product provides a unified access point to internal and external Web applications as well as portal access to other legacy applications. In this way, users sign on to the portal and receive personalized Web pages.



Figure 10-1 Horizontal and vertical portals

The personalized single point of access to all necessary resources reduces information overload, accelerates productivity, and increases Web site usage. In addition, portals do much more; for example, they provide additional valuable functions such as security, search, collaboration, and workflow.

A portal delivers integrated content and applications, plus a unified, collaborative workplace. Indeed, portals are the next-generation desktop, delivering e-business applications over the Web to all kinds of client devices.

IBM WebSphere Portal has been designed in response to the following set of fundamental business objectives:

- ▶ A single point of access to all resources associated with the portal domain
- ▶ Personalized interaction with the portal services

- ▶ Federated access to hundreds of data types and repositories, aggregated and categorized
- ▶ Collaboration technologies that bring people together
- ▶ Integration with applications and workflow systems

IBM, as well as some industry analysts, have coalesced around the concept of horizontal and vertical portals. Horizontal portals are the primary infrastructure upon which a portal is built. Vertical portals are built upon the horizontal layer and represent a specific portal instance, usually defined by a major topic or domain.

As illustrated in Figure 10-2 on page 276, the horizontal portal infrastructure consists of several modular subsystems, including the following:

- ▶ Presentation layer: A Web user interface plus pervasive device support
- ▶ Personalization: The ability to serve a dynamic response to the user based on personal profiles
- ▶ Collaboration: Tools that allow e-mail, team rooms, shared Places, and so on, to be exchanged
- ▶ Portlets: A framework for easily attaching software modules (portlets) and services
- ▶ Applications and workflow: Integration of legacy and new applications
- ▶ Search and navigation: Categorizing repositories of content and searching them for relevant information
- ▶ Publish and subscribe: The ability to author new content and publish it to subscribers
- ▶ Administration and security: Basic Web site services, such as page designers, performance monitors, cluster services, and meta data management Integration (meta data sharing, XML, connectors, standards, and EAI)

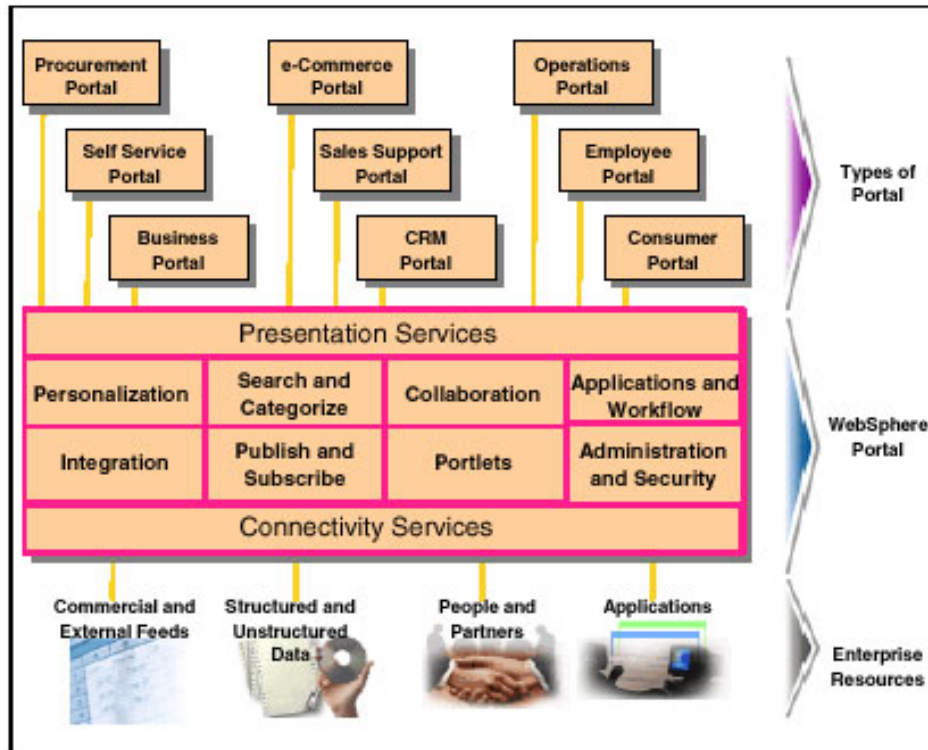


Figure 10-2 WebSphere Portal architecture

WebSphere Portal provides additional services, such as single sign-on, security, Web content publishing, search and personalization, collaboration services, enterprise application integration, support for mobile devices, and site analysis.

10.1.2 Portlets overview

Portlets are the heart of WebSphere Portal. They are small portal applications, usually depicted as a small box in the Web page. Portlets are developed, deployed, managed, and displayed independent of other portlets. Administrators and end users compose personalized portal pages by choosing and arranging portlets, resulting in customized Web pages.

However, portlets are more than simple views of existing Web content. A portlet is a complete application, following a standard model-view-controller design. Portlets have multiple states and view modes, plus event and messaging capabilities.

Portlets run inside the portlet container of the Portal Server component, similar to the way a servlet runs on an application server. The portlet container provides a run-time environment in which portlets are instantiated, used, and finally destroyed. Portlets rely on the WebSphere Portal infrastructure to access user profile information, participate in window and action events, communicate with other portlets, access remote content, look up credentials, and store persistent data.

Portlet modes

Portlet modes allow a portlet to display a different user interface, depending on the task required of the portlet. A portlet has several modes of display that can be invoked by icons on the portlet title bar: View, Help, and Edit.

A portlet is initially displayed in its View mode (Figure 10-3). As the user interacts with the portlet, the portlet may display a sequence of view states, such as forms and responses, error messages, and other application-specific states. Help mode provides user assistance. Edit mode provides a page for users to change portlet settings. For example, a weather portlet might provide an Edit page for users to specify location. Users must be logged into WebSphere Portal to access Edit mode.

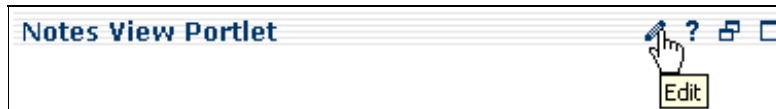


Figure 10-3 Entering Edit mode of a portlet with other “mode” icons shown

Each portlet mode can be displayed in normal, maximized, or minimized states. When a portlet is maximized, it is displayed in the entire body of a page, replacing the view of other portlets. When a portlet is minimized, only the portlet title bar is displayed on the page.

10.2 Versions of WebSphere Portal and Sametime

This section is intended to highlight an important integration consideration between Sametime and IBM WebSphere Portal Server. Depending upon which version of WebSphere Portal is purchased and deployed, this may impact whether or not the Lotus Collaborative Components API is included with WebSphere Portal. Ultimately, this also determines the API options available and the flexibility of your approach when integrating Sametime collaborative functionality into portlets.

While writing this redbook, we used the IBM WebSphere Portal Extend for Multiplatforms Version 4.2.

Note: At the time of writing, WebSphere Portal for Multiplatforms Version 5.0 was just about to be released. WebSphere Portal for Multiplatforms Version 5.0 continues the unified portal strategy for IBM and will ultimately replace WebSphere Portal for Multiplatforms Version 4.2. For additional details on Version 5.0, check the WebSphere Portal zone at:

<http://www7b.software.ibm.com/wsdd/zones/portal/>

There are two options when purchasing a WebSphere Portal Server for multiplatforms:

- ▶ IBM WebSphere Portal Enable for Multiplatforms: This offering is the basic edition of WebSphere Portal for Multiplatforms. It helps you quickly build scalable portals to simplify and accelerate access to personalized information and applications.
- ▶ IBM WebSphere Portal Extend for Multiplatforms: Includes all the robust features of WebSphere Portal Enable and introduces collaboration capabilities, enterprise search functions, and portal usage analysis. These functions help you improve employee productivity and continually strengthen relationships with your customers and trading partners.

When considering the possibility to enable your portlets with Sametime features and functionality, there is a significant difference between these two versions of WebSphere Portal.

WebSphere Portal Extend *includes* the Lotus Collaborative Components. They are UI-neutral API methods and tag libraries that allow developers who are writing portlets for WebSphere Portal to add Lotus Software collaborative functionality to their portlets. Application developers using Collaborative Components can design and implement UI extensions that leverage the features of Domino, QuickPlace, Sametime, and Lotus Discovery Server™.

WebSphere Portal Enable *does not* include all those components. That means that if you would like to Sametime-enable your portlets, you will need to use the Sametime classic APIs, namely the STLinks or Java client APIs.

10.2.1 Included collaborative portlets

While the objective of this chapter is to help you develop Sametime functionality into existing portlets, it is also helpful to understand some of the collaborative portlets that are provided with IBM's WebSphere Portal.

WebSphere Portal Extend includes a base set of collaborative portlets designed to work with Lotus Software companion products for advanced collaboration. A listing of these is shown in Figure 10-4. While a discussion of each of these portlets extends beyond the scope of this redbook, it is significant to note that several of these portlets already take advantage of integrated Sametime functionality. Several portlets include presence awareness and the ability to act upon live names with menu actions. With the introduction of Collaboration Center for WebSphere Portal, as described in 10.2.2, “Collaboration Center for WebSphere Portal” on page 279, Lotus has gone to the next level to demonstrate tightly integrated Sametime functionality designed directly into each of the three portlets.

Collaborative Portlets	
My iNotes	Provides access to a Lotus iNotes server Welcome, Mail, Calendar, To Do List, Contacts, and Notebook functions.
My Notes Calendar, Mail, and To-do List	These portlets display the user's calendar, inbox, or to-do list from their mail database.
Notes Discussion	Views Notes databases built with the Discussion Database Template.
Notes View	Views can be configured to view any Notes database.
Lotus QuickPlace	Displays a Lotus QuickPlace view inside the portlet.
Sametime Launch	Launches the Sametime Java Connect Client.
Notes Team room	Views Notes databases built with the Team Room Database Template.
Web Page	The Web Page portlet show the contents of an Internet or intranet Web page in a scrollable portlet.
Discovery Server	Searches a knowledge map and displays people and documents related to the search terms.

Figure 10-4 Collaborative Portlets included in WebSphere Portal Extend 4.2

10.2.2 Collaboration Center for WebSphere Portal

In April of 2003, IBM introduced the Collaboration Center, a set of three new portlets integrated into WebSphere Portal software, which enable users to more easily find, communicate with, and work with colleagues. The Collaboration Center portlets feature an online company directory, Web conferencing, and team workplaces, all of which have built-in people awareness and instant messaging.

While these portlets were made available for WebSphere Portal Version 4.2, they are directly integrated into WebSphere Portal Version 5.0. An overview of these portlets follows:

- ▶ **People Finder portlet:** An online company directory and organizational navigator. People Finder lets you find any employee by name and see the employee's contact information, background, areas of expertise, and context within the company's organizational chart (manager and peers).
- ▶ **My Lotus Team Workspaces (QuickPlace) portlet:** Lists your workplaces, which are provided by Lotus QuickPlace right on the portal page. You can search across all of the team workspaces you belong to, or you can quickly see what is new in a workplace, join a workplace, or create a new workplace.
- ▶ **Web Conferencing (Sametime) portlet:** Provides integrated tools for managing online meetings. From within the portlet, people can join existing online conferences, see active meetings they need to join, or schedule new meetings. And all these portlets are integrated and enabled with presence awareness, which indicates if a portal user is available for an instant messaging session. This allows you to start a chat session with someone you found through the people finder and then turn it into a Web conference, all without switching between applications. You never have to leave the portal to access applications and work with your associates, which helps you make faster and better business decisions.

Figure 10-5 on page 281 highlights the three Collaboration Center portlets working together on a single page.

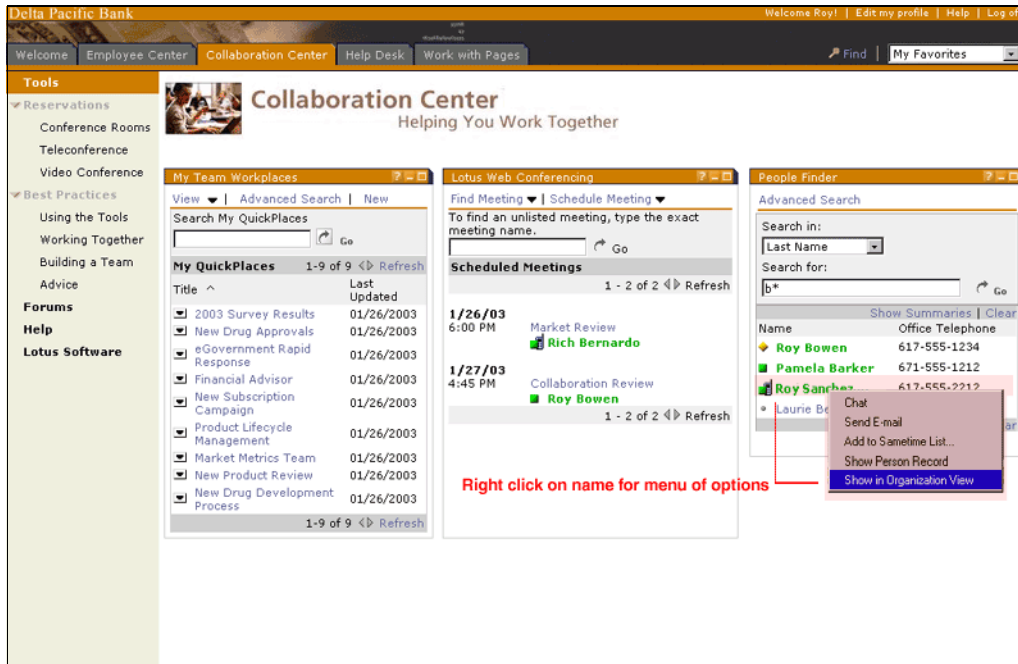


Figure 10-5 New portlets in WebSphere Version 5

10.3 Collaborative Components approach

The following sections describe the first of two suggested approaches to enabling Sametime functionality within a portlet. This first approach uses the IBM WebSphere Collaborative Components API. The second approach, described in 10.4, “STLinks API approach to enabling portlets” on page 292, achieves the same result using the Sametime Links API.

10.3.1 Using WebSphere Portal Collaborative Components

IBM WebSphere Collaborative Components are available with WebSphere Portal Extend. These are the key components that allow developers to more easily create your own collaborative portlets leveraging Lotus technologies.

This upcoming sections provide the following information:

- ▶ An overview of the Collaborative Components API capabilities
- ▶ Step-by-step introductions for Sametime-enabling basic portlet leveraging of the Collaborative Components

These sections assume that the reader has a basic understanding of the Java programming language and has developed basic Java applications.

Note: While this section serves to provide a basic overview of the functionality and capabilities of the Collaborative Components API, this subject is covered in much greater detail in the Redpaper *WebSphere Portal 4.12 Collaboration Services*, REDP0319.

10.3.2 Overview of the Collaborative Components API

The IBM WebSphere Collaborative Components API allows developers who are writing portlets for WebSphere Portal Server to easily add Lotus collaborative functionality to their portlets.

All of the Collaborative Components except for the Person and Menu tags are “UI Neutral”. In other words, the goal of most of the Collaborative Components is to provide the data from collaborative systems, and to allow the developer to execute actions on the Lotus collaborative products while also leaving the user interface up to the developer.

Collaborative Components hide the configuration details of the Lotus products that are installed in the enterprise. Developers using these components can add collaborative functionality to a portlet without regard to server configuration specifics. For example, a developer can use the people awareness tags without having to know the name of the Sametime or LDAP server.

Attention: Collaborative Components are implemented in Java and include no platform-specific code. They can be used on any J2EE-compliant server.

Types of Collaborative Components

The Collaborative Components can be divided into two main categories:

- ▶ Java Classes and Methods (cs.jar)

This package contains all the Java implementations of the collaborative components. There are classes and methods for leveraging Domino, QuickPlace®, Sametime, and Discovery Server.

- ▶ Java Script tag libraries (people.tld and menu.tld)

These tag libraries provide Sametime awareness and contextual menus to JSPs.

When to use the Collaborative Components

The goal of the Collaborative Components is to expose the most commonly used aspects of the Lotus collaborative technologies through a simple and consistent API. The components are not a replacement of the core product APIs, but rather are complementary. Developers may choose to use the Collaborative Components when they need quick and easy access to Lotus technologies, and may also use the core product APIs in other portions of their applications when more advanced integration with the Lotus Collaborative technologies is required.

10.3.3 Technical overview of the Collaborative Components

The Collaborative Components consist of infrastructure objects, service objects (and their subordinate objects), and tag language descriptors for custom tags. This section provides a high-level overview of these key objects and tags.

For more details on each class, tag, and the available methods, see the IBM WebSphere Portal V4.2 Info Center, found at:

<http://publib.boulder.ibm.com/pvc/wp/42/ext/en/InfoCenter/index.html>

Once there, look for Collaborative Components API under the Developing Portlets link.

Infrastructure objects

The infrastructure objects are as follows:

- ▶ CSEnvironment
Initializes the server environment and retrieves credentials for the logged-in user.
- ▶ CSCredentials
Represents the authentication information for the logged-in user. CSFactory uses these credentials to authenticate against the worker servers (Sametime, QuickPlace, and so on).
- ▶ CSFactory
Constructs the Java service objects and generates a connection to the appropriate worker server.

Java service objects

The CSFactory object instantiates these service objects. The developer is responsible for calling the cleanup() member function on each object before they go out of scope. This cleanup function releases any resources a service object holds.

► DominoService

The DominoService is a set of methods that provide standardized access to all versions of Domino from Domino 4.6 through 6.0, making Domino version differences transparent for portal and portlet development.

► QPService

Provides the ability to create a QuickPlace.

► PeopleService

The PeopleService retrieves information about a specified person from the directory configured to work with Portal Server.

► DiscoveryServerService

Provides the ability to discover, search for, and retrieve Categories, People, Places, and Documents from a Discovery Server.

Tag language descriptors for custom tags

Collaborative Components include tag language descriptors (TLDs) for custom People and Menu tags that can be used with certain service objects.

► PeopleService tags

The people service tags generate context sensitive menus and Sametime awareness for names found in the LDAP directory. It contains tag language descriptors for the following tags:

– person

Enables collaboration on the enclosed name. If Sametime is installed and enabled to work with the Portal Server, then the person tag also provides online status for the person link and the following additional actions on the person menu:

- Chat
- Share
- Available Tools
- Add to Sametime List

– peopleinit

Determines whether Sametime or Discovery Server, or both, are enabled to work with Portal Server and generates the correct HTML and JavaScript for initializing Sametime or enabling Discovery Server, or both. Establishes the server connections between the Portal Server, the Sametime server, and the Discovery Server, and provides automatic log-in to all servers.

Important: The `peopleinit` tag is included in the banner (`banner.jsp`) of a collaborative portal; it is not for use inside any portlets.

- `peopleend`

Finalizes the `people` tag functionality. Do not use this function inside a portlet, as it is initialized by the portal server itself at startup.

► **Menu tags**

The Menu tags allow developers to add menus to their Collaborative Components applications. The `PeopleService` tags use the Menu tags to display person link menus next to a person's name.

- `menuinit`: Initializes the menu service.
- `menucontext`: Allows a set of menus to be defined. May also hold a parameter specified by the `menudisplay` tag.
- `menu`: Defines a menu.
- `menuitem`: Defines a menu item, its name, and associated Java Script callback function. The Resource service may be used here to internationalize menu items. It offers `enable` (black/gray) and `show` (show/hide) optional attributes.
- `menudisplay`: Shows the menu. Through the use of `paramName/paramValue` attributes, sets a context variable for use in callback functions.

10.3.4 Adding Sametime collaboration to a basic portlet

This section describes how to use the Collaborative Components API described earlier in this chapter to add Sametime features to an existing portlet.

Attention: To accomplish the tasks described here, we assume that a developer has some basic portal administrative knowledge (installing portlets), as well as some basic skills using IBM WebSphere Studio Application Developer. Please, refer to the Redpaper *WebSphere Portal 4.12 Collaboration Services*, REDP0319 in order to see step by step instructions on how to use WebSphere Studio Application Developer to accomplish these tasks.

Basic portlet example: Name List portlet

For the purpose of illustration, we have chosen a basic portlet that displays a listing of names from a corporate directory. This sample portlet is called Name List, and can be downloaded from the Redbooks Web site. See Appendix C,

“Additional material” on page 467 for more detailed instructions on how to download this sample portlet.

Figure 10-6 shows the portlet in view mode. It simply displays a list of names from a directory, but does not yet provide either presence awareness, or the ability to act upon a name.

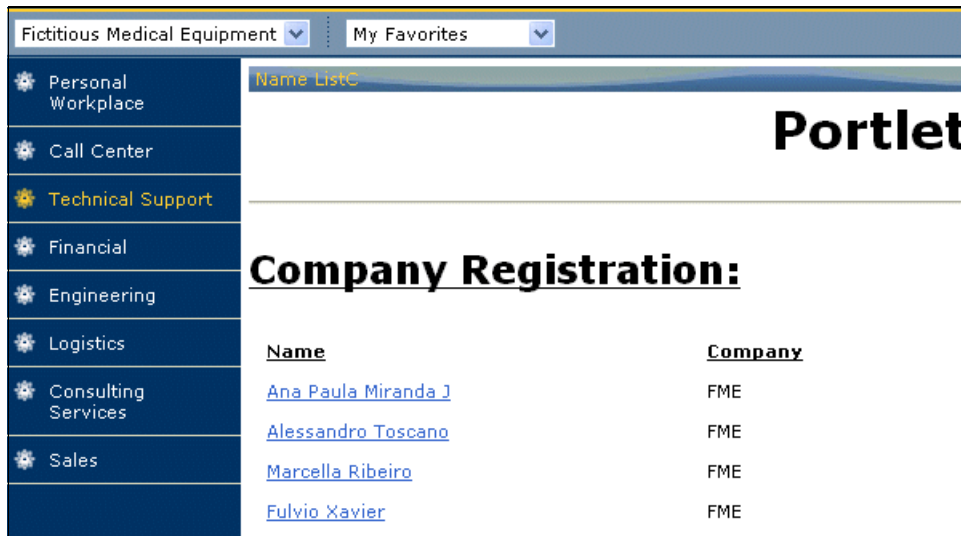


Figure 10-6 Simple portlet example: names are not yet Sametime enabled

In the following sections, we use the Sametime features within the Collaborative Components API to add awareness to these names.

Important: The source code for the portlet shown on Figure 10-6 on page 286 is available as additional material in this redbook. Please refer to Appendix C, “Additional material” on page 467 for detailed instructions on downloading this code sample.

10.3.5 Adding awareness to the JSP

At this point, we will implement the Collaborative Components resource to add awareness to the JSP pages that makes this portlet.

To accomplish this, we first need to add the resources shown in Table 10-1 on page 287 to the portlet project in WebSphere Studio Application Developer.

Table 10-1 Resources to be added to NameList portlet project

Resource	Source location	Destination
cs.jar	<wsroot>\AppServer\lib\app\cs.jar	/NameList/webApplication/WEB-INF/classes
people.tld	<wsroot>\PortalServer\app\wps.ear\wps.war\WEB-INF\tld\people.tld	/NameList/webApplication/WEB-INF/tld

With these files available within the project, we can now work with and modify the View.JSP file.

Within the <HEAD> tag of the View.JSP file, we must point the taglib directive to the people.tld file and import the required packages, as shown in Example 10-1.

Example 10-1 Pointing the taglib and importing required packages

```
<HEAD>
...

<%@taglib uri="/WEB-INF/tld/people.tld" prefix="p"%>
<%@page language="java" import="com.lotus.cs.*" session="true"
isThreadSafe="true" isErrorPage="false"%>

...
</HEAD>
```

With the people tag library (people.tld) properly referenced and the collaboration services classes (cs.jar) imported, now we must add the proper tags to the JSP file in order to enable names awareness.

Within the View.jsp file, locate the following line:

```
<%= (String) NameListPortletBean.getNamePrefix().elementAt(i) %>
```

This line writes the people names within an HTML page in order to give the result shown in Figure 10-6 on page 286. To enable people awareness to the View.jsp file, we just need to include the <p:person> tag to this line:

```
<p:person><%= (String) NameListPortletBean.getNamePrefix().elementAt(i)
%></p:person>
```

Finally, save the changes to the View.jsp file, and we are ready to deploy the Sametime-enabled NameList portlet.

Once deployed, the Sametime-enabled NameList portlet should show “live names” for those people who are also logged into the server. As shown in Figure 10-7 on page 288, the name for Ana Paula Miranda is now live.

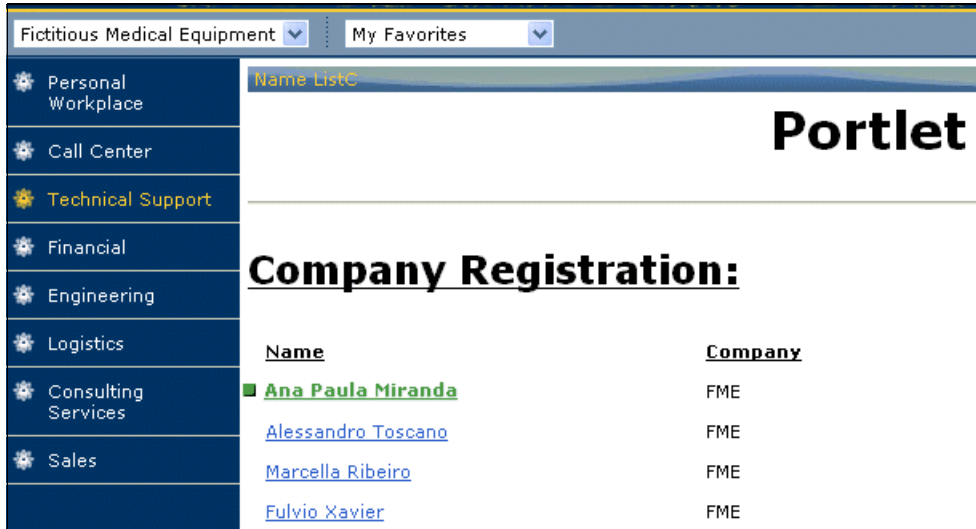


Figure 10-7 Sametime enabled portlet example

Here we have illustrated how by adding 2 lines to the <HEAD> tag and by adding one tag around the person name, we added an online collaboration to our portlet. Instead of having just list of black names, we now can have live names with their statuses. The person tag also adds the “chat” option automatically to the person menu, as you can see in Figure 10-8.

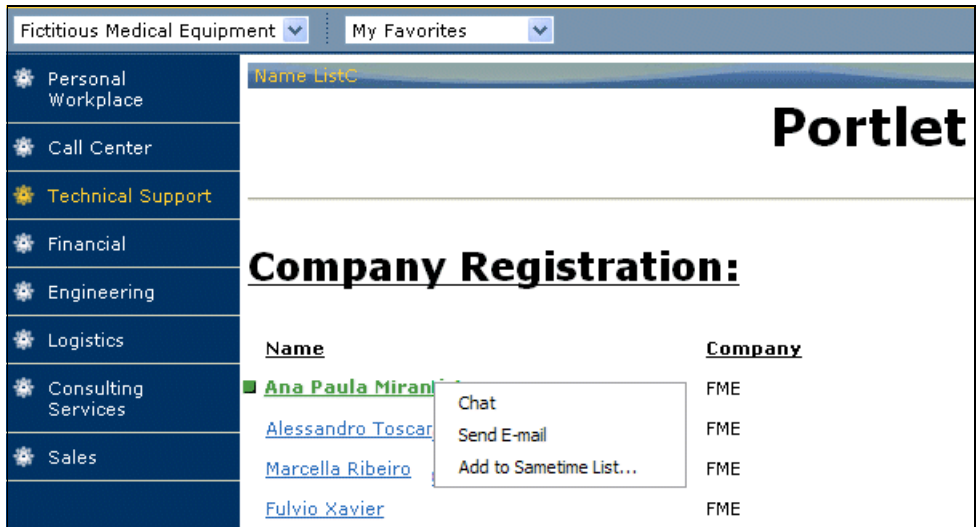


Figure 10-8 The chat menu when clicking on a live person

Using the person tag eliminates the need to worry about logging in to the Sametime server and providing the server with authentication information. The tag does everything for you with only minor changes to the code.

10.3.6 Adding more Sametime functionality

Until now, it was very easy and simple to add awareness and chat to your portlet. But what if you want to do more? For example, what if you would like to change your status directly from within the portlet? How can we do it? The person tag will alone will not be sufficient to help us in this case.

As we will demonstrate in the next section, this is also a relatively easy function to build in. All you need to do is use any of the STLinks API calls, which are made available to you, since you included the people tag library (people.tld) and the collaboration services classes (cs.jar file) in Example 10-1 on page 287. It is actually the case that the person tag is already using the Sametime STLinks API, so all of what you need is already included. You only need to add your specific calls.

Adding an I'm Away button to our portlet

Within this next example, we will demonstrate how to change your online status directly from within the portlet. We will add the ability to change the status from active to away, and also provide the ability to directly edit the away message associated with the status. The end result will provide a button and editable text field similar to the one shown in Figure 10-9 on page 290.



Figure 10-9 Adding a button to change user status

First we will add a I'm Away button and a text field for the status description to the page. To accomplish this, we will add the lines in Example 10-2 to the file View.JSP after the table tag.

Example 10-2 Code for I'm Away button

```
<form name="f1">
<input type=button value="I'm Away" onClick=changeMyStatusToAway()>
<input style="width:30%" name=desc>
</form>
```

With this modification, a button and editable text field is added to the portlet. When the user clicks on the I'm Away button, the user's status is changed to Away and their status description will display the text they entered in the text field.

Next, we add the functional code (Example 10-3 on page 291) to define the click event of the button by implementing the changeMyStatusToAway function.

Example 10-3 Code for changeMyStatusToAway function

```
<script>
function changeMyStatusToAway()
{
    STLinksSetMyStatus(64,document.f1.desc.value);
}
</script>
```

Once these two changes have been completed, the button should be functional and ready for testing. Figure 10-10 illustrates our portlet with the new stlinks functionality, namely the I'm Away button. Notice that once the user set their status to Away, the connect client also displays the user in away mode.



Figure 10-10 Changing status through the I'm Away button

The ability to use STLinks API calls is provided automatically, since we already added the taglib include (people.tld file) and cs.jar file include to our page. By including this ability, it also provides functions for connecting us to the Sametime server using the portal authentication information. It also references the include for the stlinks.js file that contains all the STLinks API calls.

In this example, we have used the STLinksSetMyStatus API call to provide the functionality for the button. Please be aware, however, that this is just one of the many STLinks API calls that is available to a developer. For a complete listing of the STLinks API function calls available, please reference the Developers Guide provided with the Sametime Links Toolkits.

10.4 STLinks API approach to enabling portlets

As mentioned at the beginning of this chapter, there are two approaches to accomplish the same result for Sametime-enabling portlets. Using the Collaborative Components API provided with WebSphere Portal represents one approach. We will now discuss an alternative approach which may prove desirable for a variety of reasons. Continuing with the same basic portlet, Name List, this section describes how to achieve the same result using a “pure” Sametime Links approach.

10.4.1 Why use the STLinks API

Prior to showing how to implement the Sametime functionality, it is important to understand when and why the STLinks API approach to enabling portlets may be the best solution.

There are two key considerations:

- ▶ Is your company using IBM WebSphere Portal? If so, and assuming that the version they are running is either Portal Server Extend or Experience, then the Collaborative Components will be available. If your company is *not* using IBM WebSphere Portal, but is instead running an alternative portal technology, the Collaborative Components API is not included. In this case, the STLinks API provides a solution for Sametime-enabling portlets.
- ▶ Size and performance considerations. If performance of the required loading time for portlets will be a significant issue, then using the STLinks API presents advantages. The `cs.jar` used by the Collaborative Components is around 180 KB, while the `stlinks.jar` is around 30 KB.

10.4.2 Enabling the portlet using STLinks

In order to use Sametime Links, all you have to do is make your JSP page write the Sametime Links required tags and JavaScript functions.

To show you how this same task can be performed using Sametime Links, we will reproduce live names for the Name List portlet. This is the same example as the one shown in 10.3.5, “Adding awareness to the JSP” on page 286. This functionality is also discussed in 9.3, “Enabling live names in a Web page” on page 223.

Making the names live

The HEAD tag of an JSP page is exactly the same as any HTML page. Hence, we need to add the code to set required parameters to call Sametime Links, as shown in Example 10-4 on page 293.

Example 10-4 JSP HEAD tags to call Sametime Links

```
<HEAD>
...
<LINK REL=STYLESHEET HREF="http://<st server>/sametime/stlinks/stlinks.css"
TYPE="text/css">

<SCRIPT src="http://<st server>/sametime/stlinks/stlinks.js"></SCRIPT>

<SCRIPT>
    setSTLinksURL("http://<st server>/sametime/stlinks","en",
        "http://<st server>/sametime/stlinks")
</script>
...
</HEAD>
```

Note that the `<st server>` in Example 10-4 stands for the host name of the Sametime server used for enabling the application.

Next, we need to call the Sametime Links applet passing the necessary login information, as shown in Example 10-5.

Example 10-5 Call to the Sametime Links Applet with logon information

```
<script>
    writeSTLinksApplet("<user name>","<password>",<is by token flag>);
</script>
```

By integrating with your Web application server, the user name and password properties can be passed as a token. Without this, the user needs to pass the login information again.

Finally, within the JSP page we are working on, we must add the `writeSametimeLink()` function where necessary in order to present names with Sametime awareness.

Find the following code within the JSP page:

```
<%= (String) NameListPortletBean.getNamePrefix().elementAt(i)%>
```

Rearrange this code in order to get the results shown in Example 10-6.

Example 10-6 writeSametimeLink() function within a JSP page

```
<script>
    writeSametimeLink(
        "<%= (String) NameListPortletBean.getNamePrefix().elementAt(i)%>",
        "<%= (String) NameListPortletBean.getNamePrefix().elementAt(i)%>",
        false);
```

</script>

By using this technique, we achieve the same result shown in Figure 10-7 on page 288.

Adding the I'm Away button

Regarding the I'm Away button, you can add the same code as we demonstrated in Example 10-2 on page 290 to the JSP page, and it will work the same way. You do not need to change any of this code, because you will recall that this code already uses the STLinks API itself and not the Collaborative components tags. Simply follow the same steps as those outlined in 10.3.6, "Adding more Sametime functionality" on page 289 to recreate the button and custom text fields.

Adding custom pop-up menus

Finally, there is the remaining issue of how to add the customized pop-up menu options from the live names. When using the Collaborative Components approach, the person tag (peple.tld file) automatically provided this functionality. When using the STLinks API approach, however, it is necessary to build this functionality in.

Fortunately, this topic of adding custom pop-up menu options to names using Sametime Links has been covered in significant detail in 9.7, "Adding menu options to Sametime Links" on page 259. Please refer to this section for details on how to add any of the following menu options from a name within the page:

- ▶ **Message**
Sends an instant message to the selected person.
- ▶ **Share Application**
Invites the selected person to an application-sharing meeting.
- ▶ **Send Mail**
Creates a new e-mail message for the selected person with the user's default e-mail client.
- ▶ **Add to Contact List**
Adds the selected person to the user's Sametime Connect's contact list.

10.4.3 Conclusion

This chapter has illustrated how to add Sametime functionality to a portlet, using either the WebSphere Collaborative Components API or using an approach based on Sametime Links. While the collaborative components provided with

WebSphere Portal Server are convenient, the Sametime Links approach provides an alternative approach for organizations who are not using WebSphere Portal Server for the foundation of their portal framework.



Customizing the Online Meeting Center

The Sametime Online Meeting Center (Meeting Center) is the entry point for user scheduling and attending scheduled Sametime meetings. It is a Web-based Domino application that creates document for every scheduled meeting. While the look and feel and features of the Meeting Center are functional, many find they are interested in changing or tweaking the look and feel of Meeting Center.

In this chapter, we will discuss:

- ▶ Typical reasons why people brand the Meeting Center
- ▶ An example of branding the Meeting Center
- ▶ Sending an e-mail notification whenever a meeting is created, updated, or deleted.

11.1 Typical reasons for branding the Meeting Center

Users schedule, find, and attend meetings by accessing the Meeting Center through a Web browser. When they do, more often than not, the transition from their company designed Web pages to the Meeting Center start page is awkward and/or confusing because the look and feel of the two pages is usually completely different. The colors are different. The page logo is different. The page navigation and menus are different. This is because the Meeting Center comes branded with IBM Lotus colors and logos. Many would like to change these colors and logos to provide a more seamless transition between other company Web pages in their Web sites and the Meeting Center, and comply with their company's design standards.

Important: According to Technote # 1101472 (<http://www-1.ibm.com/support/docview.wss?uid=swg21101472>), changes to the Sametime Meeting Center database are not supported by Lotus Support. The Technote goes on to say that “certain changes can conflict with exist code, hidden or visible.”

Important: Any design changes made to any of the Sametime templates or databases will need to be re-applied each time you upgrade the Sametime server.

11.2 Branding the Meeting Center

There are many different ways to customize the Meeting Center, depending on your company's business needs, preferences and infrastructure. Since it is a Domino database, you can do many of the same customizations you might consider for other Domino databases, such as inserting your company's logo, color schemes, or standard navigation links. You can also front-end it using Java Server Pages (JSP), Active Server Pages (ASP), or Java servlets. Instead of trying to cover all the different types of customizations you might consider, this section walks through an example that focuses on common customizations, such as changing the page header, background, and some minor style tweaks.

Figure 11-1 on page 299 shows the out-of-the-box (or before) look and feel of the Meeting Center.

Lotus Sametime
Meeting Center

Wed, 7/2/2003 3:04 PM
Logged on as Jonas Covey

[New Meeting](#)

Active

- Scheduled

[Finished](#)
[Today](#)

[Unlisted Meeting](#)
[All Meetings](#)
[My Meetings](#)
[Recorded](#)
[Calendar](#)

[Test Meeting](#)
[Help](#)

Scheduled Meetings

Click on a meeting name to see the Meeting Details.

Date	Meeting	Moderator	Status
7/3/2003	3:00 PM Chapter 1 Discussion	Jonas Covey	Scheduled
7/7/2003	3:00 PM Weekly Status Update	John Bergland	Scheduled
7/8/2003	3:15 PM Sametime Link Review	Carl Tyler	Scheduled
8/4/2003	3:00 PM Monthly Sales Update	washington cabral	Scheduled
9/1/2003	8:00 AM Performance Review	John Barrow	Scheduled
10/6/2003	3:00 PM Monthly Sales Update	washington cabral	Scheduled

Figure 11-1 Before: Sametime Meeting Center Out-of-the-box

Figure 11-2 on page 300 shows our customized Sametime Meeting Center.

e-Meetings

[directory](#)
[search](#)
[faq](#)
[Feedback](#)

New Meeting

[Active](#)
[Scheduled](#)
[Finished](#)
[Today](#)

[Unlisted Meeting](#)
[All Meetings](#)
[My Meetings](#)
[Recorded](#)
[Calendar](#)

[Test Meeting](#)
[Help](#)

Scheduled Meetings

Click on a meeting name to see the Meeting Details.

Date	Meeting	Moderator	Status
7/6/2003	11:45 AM Free Jam	Jonas Covey	Scheduled
7/7/2003	3:00 PM Weekly Status Update	John Bergland	Scheduled
7/8/2003	3:15 PM Sametime Link Review	Carl Tyler	Scheduled

[Previous](#) | [Next](#)

e-Meetings

[directory](#)
[search](#)
[faq](#)
[Feedback](#)

New Meeting

[Active](#)
[Scheduled](#)
[Finished](#)
[Today](#)

[Unlisted Meeting](#)
[All Meetings](#)
[My Meetings](#)
[Recorded](#)
[Calendar](#)

[Test Meeting](#)
[Help](#)

Free Jam

Meeting Details

[Edit Meeting...](#)
[Delete Meeting...](#)

Essentials

Meeting Name	Free Jam
Start time	Fri, 7/4/2003 11:45 AM
Duration	0d 1h 00m
Has password	No
Status	Scheduled
Moderator	Jonas Covey
Encrypted	Yes

Repeating	Repeat Start Date
Daily	7/4/2003
Every day	For
	3 Days
	At Weekends:
	Don't Move

Locations

Participants within the organization can attend from the following servers

<http://itsotest-st31.cam.itso.ibm.com/stconf.nsf/meeting/d39f3736b639e47685256d58005559f7>

Figure 11-2 After: Customized Sametime Meeting Center

Important: The Meeting Center (stconf.nsf) inherits its design from the Sametime Online Meeting Center Template (stconf31.ntf). All design changes should be made to the template. The design element should also be signed with the appropriate Notes ID before being deployed.

Also, do not use the Notes 6 client on a Notes 5 database.

11.2.1 Changing the page header

The page header section for each view and form is loaded from the WebTitleLogin subform. Changing this subform is simple because it only contains pass-thru HTML and one single JavaScript function to force a login to the database.

Here we replaced the existing pass-thru HTML with our own HTML. We included the company logo and some standard navigation links.

Modifying this subform will change the page headers for most of the pages in the database except for those pages that are programmatically generated. The following is a list of design elements containing the page header that are programmatically generate:

- ▶ Agents\(\WebReplaceRecording)\PrintHTMLMsg (Script)
- ▶ Agents\(\WebImportRecording)\PrintHTMLMsg (Script)
- ▶ Script Libraries\ConferenceLibrary\PrintHTMLLoginPage (Script)
- ▶ Agents\(\HiddenConferenceSave) (Script)
- ▶ Script Libraries\ConferenceLibrary\PrintHTMLPage
- ▶ Forms\\$\$ReturnAuthorizationFailure (Pass-thru HTML section)
- ▶ Forms\\$\$ReturnGeneralError (Pass-thru HTML section)
- ▶ Forms\\$\$ReturnAuthenticationFailure (Pass-thru HTML section)

Tip: Instead of inserting pass-thru HTML into the subform, use a computed-for-display field that does a lookup to a configuration document that contains the page header HTML. This approach would make the page header easier to integrate into the programmatically generated pages.

Tip: If you are only interested in swapping your logo for the Sametime logo, follow these steps:

1. Create an GIF file named stwordmark.gif that is 155 x 26 pixels.
2. Make a backup of <html>\sametime\images\stwordmark.gif from the Sametime Server.
3. Replace the GIF backed up in Step 2 with the GIF created in Step 1.
4. Clear your browser cache and test.

11.2.2 Changing the look

Changing the page backgrounds, colors, and fonts requires a little more work than changing the page header, because you need to change several design elements instead of just one. The primary reason for this is because the Meeting Center database inserts the same in-line stylesheets in several design elements. In addition, styles were defined in other areas of the code, such as in the HTML body tag. Instead of continuing with this design pattern, we choose to try to consolidate all these styles into one stylesheet that would also include some custom styles. Example 11-1 shows the stylesheet that we used. The stylesheet did not fix everything, because a lot of styling is still embedded in-line in the application.

Example 11-1 Custom Meeting Center stylesheet

```
body {
    margin-top: 0;
    margin-left: 0;
}
span, p, ul, ol, td, th, tr, blockquote {font-size:10pt; font-family:
verdana,helvetica,arial,sans-serif}
a:hover {color:red}
a:active {color:#black}

.f {color: blue; font-weight:bold}
.f a:link {color: black}
.f a:visited {color:black}
.f a:hover {color:red}
.f a:active {color:black}

.btn-66f, .btn-999, .btn-666, .btn-fff {font-weight: bold; font-size:8pt;}
.btn-66f, .btn-66f a:link, .btn-66f a:active, .btn-66f a:visited, .btn-66f
a:hover {color:#6666ff; text-decoration:none}
        .btn-66f a:hover {color:#9999ff}
        .btn-66f a:active {color:#3333cc}
```

Next, we removed all the embedded in-line stylesheets references. It is best to start with the `$$ViewTemplate` forms. All public views are displayed through a `$$ViewTemplate` form. Most views have a corresponding `$$ViewTemplate` form. Those that do not are displayed through the `$$ViewTemplateDefault` form. In each `$$ViewTemplate`, we performed the following tasks:

- In the HTML Head Content section, remove the inline stylesheet text:

```
+ @NewLine +
"<!--" + @NewLine +
"BODY { font-family: Arial, Helvetica, sans-serif; " + @NewLine +
" background-color: #FFCC00; " + @NewLine +
" background-image: URL(/sametime/images/stbackground.gif); " + @NewLine +
```



```
" background-repeat: no-repeat; " + @NewLine +
"}" + @NewLine +
"TR, TD, TH { font-family: Arial, Helvetica, sans-serif; }" + @NewLine +
"-->" + @NewLine +
"</style>"
```

- ▶ In the HTML Head Content section, we inserted a reference to the stylesheet. For example, we inserted the following:

```
<link rel="stylesheet" type="text/css"
href="/sametime/custom/styles/stconf.css">
```

- ▶ In the HTML attributes section, we removed the link, vlink, and alink attributes. In most cases, only "> <!-- <ignoreform" is remaining, which must be present in order for the application for run properly.

We performed the preceding tasks on the following `$$ViewTemplate` forms:

- ▶ `$$ViewTemplate` for `vwWebRecordedMeetings`
- ▶ `$$ViewTemplate` for `vwWebMyMeetings`
- ▶ `$$ViewTemplate` for `vwWebActiveMeetings`
- ▶ `$$ViewTemplate` for `vwWebScheduledMeetings`
- ▶ `$$ViewTemplate` for `vwWebFinishedMeetings`
- ▶ `$$ViewTemplate` for `vwWebAdminAllMeetings`
- ▶ `$$ViewTemplate` for `vwWebAdminHiddenMeetings`
- ▶ `$$ViewTemplate` for `($WebSearch)`
- ▶ `$ViewTemplate` for `vwWebTodaysMeetings`
- ▶ `$$ViewTemplate` for `vwCalendar`
- ▶ `$$ViewTemplate` for `vwWebAllMeetings`

Now that all the `$$ViewTemplates` have been modified, we move on to changing the WebConference form, which is used for creating, viewing and editing meetings. In the WebConference form, we performed the following tasks:

- ▶ In the HTML Head Content section, remove the inline stylesheet text:

```
"<style type=\"text/css\">" + @NewLine +
"<!--" + @NewLine +
"BODY { " + @NewLine +
" background-color: #FFCC00; " + @NewLine +
" background-image: URL(/sametime/images/stbackground.gif); " + @NewLine +
" background-repeat: no-repeat; " + @NewLine +
}" + @NewLine +
tmpFont + @NewLine +
"-->" + @NewLine +
"</style>" + @NewLine +
```

- ▶ In the HTML Head Content section, we inserted a reference to the stylesheet. For example, we inserted the following:

```
<link rel="stylesheet" type="text/css"
href="/sametime/custom/styles/stconf.css">
```

- ▶ In the HTML body attributes section, we removed all instances of the following text:

```
link="#"990000\" vlink="#"990000\"
```

Modifying these forms corrected all of the in-line stylesheet issues that we were concerned with except for the pages that are programmatically generated. The following is a list of design elements that programmatically insert the in-line stylesheet:

- ▶ Script Libraries\ConferenceLibrary\Subs\PrintHTMLPage
- ▶ Script Libraries\ConferenceLibrary\Subs\PrintHTMLErrorPage
- ▶ Agents\WebImportRecording\Subs\PrintHTMLMsg
- ▶ Agents\HiddenConferenceSave\Subs>ListRoutine
- ▶ Agents\WebReplaceRecording\Subs\PrintHTMLMsg
- ▶ Forms\WebAttend
- ▶ Forms\\$\$ReturnAuthenticationFailure
- ▶ Forms\\$\$ReturnAuthorizationFailure
- ▶ Forms\\$\$ReturnGeneralError
- ▶ Help About\Help About Document\Text\1. Pass-thru HTML

Tip: To change just the page background of all forms and views in the Meeting Center database, follow these steps:

1. Create an GIF file named stbackground.gif that is 845 x 677 pixels.
2. Make a backup of <html>\sametime\images\stbackground.gif from the Sametime Server.
3. Replace the GIF file backed up in Step 2 with the GIF file created in Step 1.
4. Clear your browser cache and test.

The primary drawback to the Meeting Center implementation is that the page background color is referenced using in-line stylesheets in all \$\$ViewTemplate forms and the WebConference form. Therefore, even though you change the background GIF you are still stuck with the Lotus yellow (#FFCC00) background color.

11.2.3 Using a Java Server Page (JSP) front-end

Another possible, less intrusive approach, is to use Java Server Pages (JSP) to create a front-end to the Meeting Center views (that is, Active, Scheduled, and so on) and then customize the WebConference form as described in the previous section. When using this approach, fewer design elements are modified and you can have greater control of the page layout.

The trick to this approach is that you not only need to pull in the data from the view but also all the JavaScript, computed fields, and WebQueryOpen generated code that is produced by the `$$ViewTemplate` for the view when it is displayed.

For this example, we created a JSP page that displays an active and scheduled meeting. When you click on the meeting link, the meeting document is displayed from the Meeting Center database using WebConference form (as we modified it in the previous section).

Here are the basic steps:

1. Create a new view in the Meeting Center template that lists all active and scheduled meetings.
2. Modify the Meeting column in view to generate a absolute URL to the meeting document.
3. Create a new `$$ViewTemplate` form for displaying the view created in Step 1.
4. Create a JSP page that uses the Java `URLConnection` class to read in the view created in Step 1.

A working example can be found in the code download.

Note: In the working example, we removed the design elements that are supported for A/V and recording meetings because we could not allocate enough time for testing.

Note: IBM used a similar approach when integrating the Meeting Center into their e-Meeting Web conferencing Web site.

11.3 Meeting summary e-mail

With the Notes 6 client, the Notes 6 mail template, and Sametime 3.x, users have the ability to schedule online Sametime meetings directly from a mail database. But until your company has upgraded to these minimum requirements, the following example may help you build a stop gap solution. This section describes

how to add a feature that allows users who are scheduling, updating, or cancelling a meeting to send an e-mail notification to selected participants.

Note: This example only sends out an e-mail. It does *not* create a calendar entry.

11.3.1 Overview

Once the customizations have been applied, the user can choose whether or not they would like to send an e-mail notification to the selected meeting participants when they create, update, or delete a meeting. The generated e-mail looks similar to the sample in Figure 11-3. It includes all the meetings details, including a list of the meeting attachments.

Message from the Moderator
This is a sample email

Essentials
Meeting Name: Email Example
Start Time: 07/17/2003 11:00:00 AM EDT
Duration: 0d 1h 00m
Moderator: Jonas Covey
Encrypted: Yes

Locations
Attend the meeting on your home Sametime server if it is listed below
<http://itsotest-st31.cam.itso.ibm.com/stconf.nsf/meeting/DDEE6AA63F45877185256D660050232E?OpenDocument>

Tools
Whiteboard
Meeting Room Chat
Send Web page
Computer Audio (Ensure that your computer supports audio.)

Figure 11-3 Sample e-mail

To help you visualize the customizations we will be describing, the following steps walk you through creating a new meeting and sending an e-mail notification:

1. In the Sametime Online Meeting Center, click the **New Meeting** link on the left-side navigation, fill in the meeting details, and then click the **Save** button. The e-mail notification alert box will appear, as shown in Figure 11-4.

Essentials | Files | Security | Tools | Locations

To create a meeting, complete the information below.

Meeting type: Collaboration

Meeting name:

Moderator: Jonas Covey Change...

☐ Start Now
☒ Schedule

Date (mm/dd/yyyy): << < 7/17/2003 > >> Repeat...

Time: << < 11:45 AM > >>

Duration: << < 0d 1h 00m > >>

☐ Record this meeting so that others can replay it later.

Save Cancel

Figure 11-4 E-mail notification prompt

2. The e-mail notification alert box prompts the user whether or not they would like to send an e-mail notification. Next, assuming the user clicks **OK**, the email notification form is displayed as shown in Figure 11-5 on page 308.

Meeting Notice
Use this form to send an e-mail notification with the meeting details to selected users. The essential meeting details such as start time, location, and password, are automatically included in the message.

To:

cc: Jonas Covey

Subject:

Message:

Figure 11-5 E-mail notification form

3. Fill in the e-mail notification form. To address the e-mail, click the **Add or Remove People** button. (If the meeting was restricted, the To field will automatically be populated with the restrict list.) If the **Add or Remove People** button was clicked, the standard Sametime directory applet will pop up, as shown in Figure 11-6 on page 309.

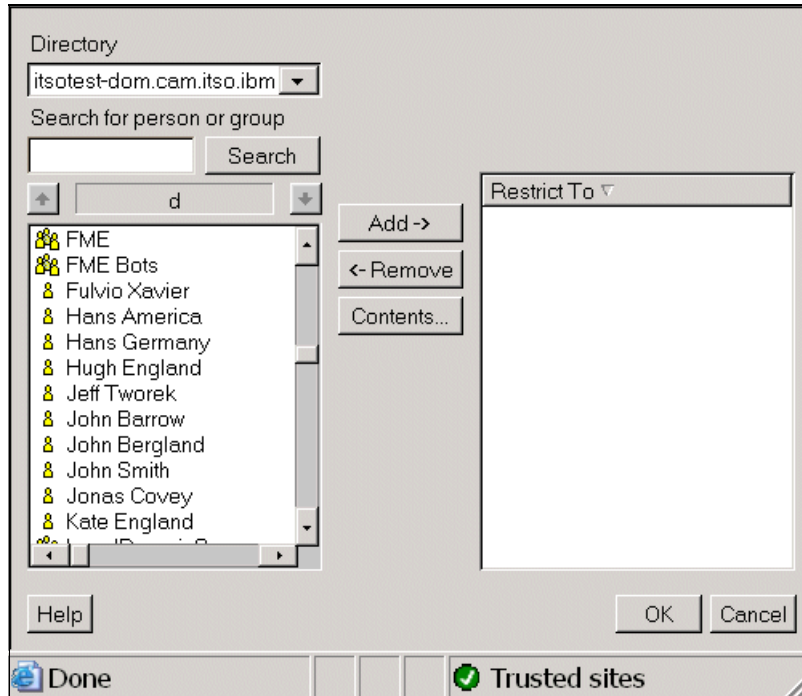


Figure 11-6 Directory applet

Select the users you would like to notify and then click the **OK** button. This will return you to the e-mail notification form. Click the **Send** button.

4. After clicking the **Send** button, an e-mail is sent to each user that was listed in the To field and a copy is sent to the creator and/or moderator.

11.3.2 Behind the scenes

While not necessary for applying the customizations discussed in the chapter, it is useful to understand what is going on behind the scenes and what Domino design elements are involved. The following diagrams (Figure 11-7 on page 310 thorough Figure 11-9 on page 312) depict which design elements are used and in what sequence when executing some basic use cases.

Creating a meeting

The following sequence of events occur when a meeting is created.

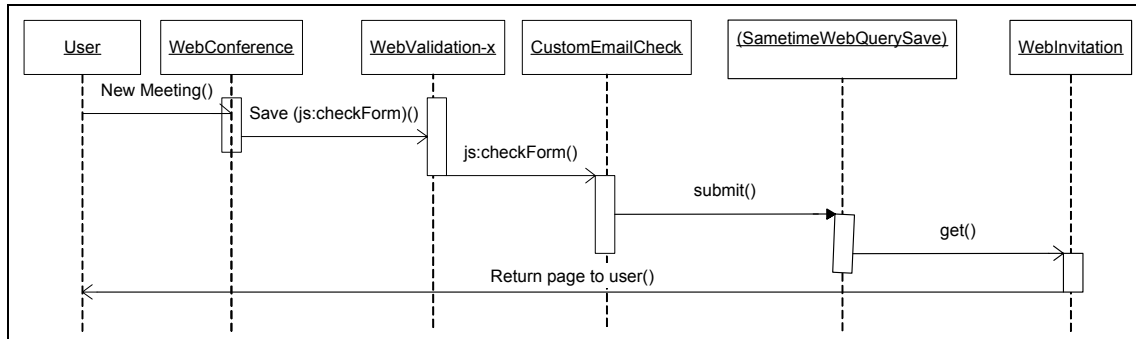


Figure 11-7 Sequence diagram: creating a meeting

A user clicks the New Meeting link and the WebConference form is displayed. Fill in the form. When finished filling out the form, the **Save** button is clicked, which triggers the checkForm() JavaScript function. The JavaScript function performs basic error checking and then executes JavaScript, which is contained in the CustomEmailCheck subform. Once these sections of JavaScript have been executed, the page is submitted to the Domino server using the HTML form.submit() method. When the form is submitted to the Domino server, the (SametimeWebQuerySave) agent is executed. This agent, among other things, determines whether or not the user has requested to send an e-mail notification. If so, the WebInvitation form is returned to the user. This form inherits field values from the meeting document.

Sending a notification

The following sequence of events occurs when a sending a notification.

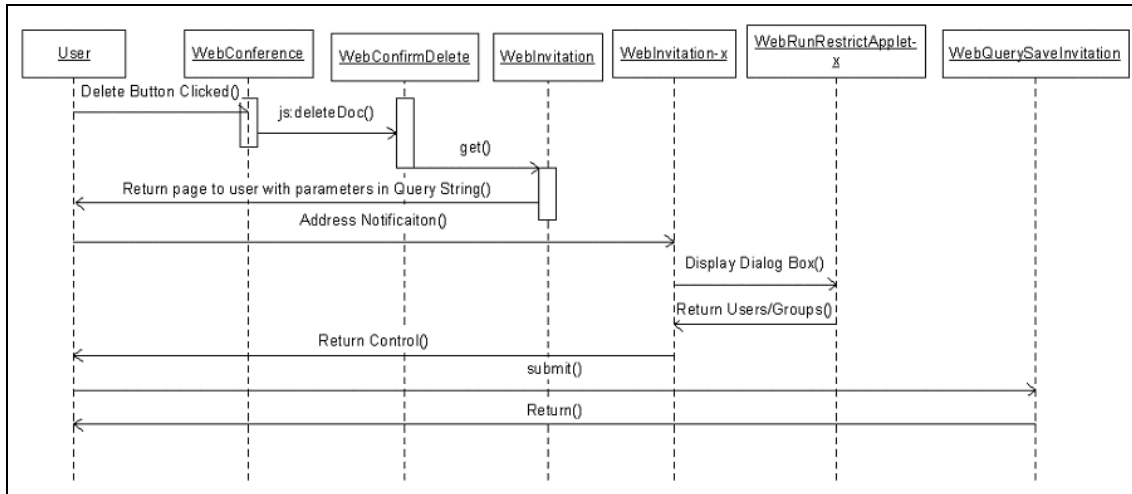


Figure 11-8 Sequence diagram: sending an invitation

The user fills in the WebInvitation forms and clicks the **Add or Remove People** button. This button is actually inserted on the form via a WebInvitation-x subform. When clicked, the WebRunRestrictApplet-x subform is loaded in its own browser window. The user selected users or groups from the directory and clicks the **OK** button. When the **OK** button is clicked, the values are returned and inserted into the WebInvitation form.

Once the user has completely filled out the form, they click the **Send** button. The form is submitted to the Domino server and the (WebQuerySaveInvitation) agent is executed. This agent sends the e-mail notification and then returns the user to the meeting page of the meeting.

Note: The e-mail notification is sent when the WebQuerySaveInvitation agent is executed.

Cancelling a meeting

The following sequence of events occurs when a meeting is deleted.

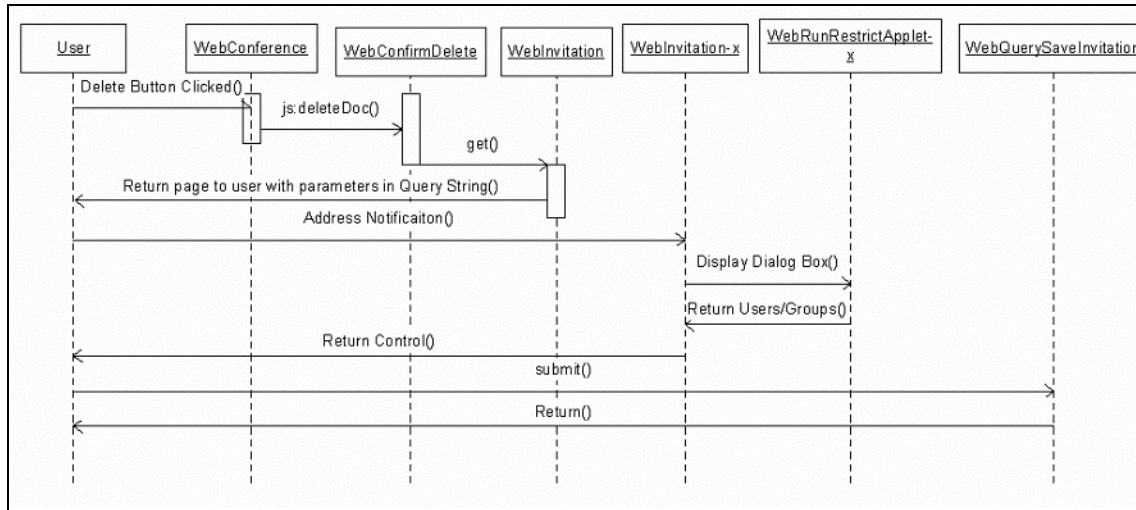


Figure 11-9 Sequence diagram: cancelling a meeting

User opens a meeting document/page is read mode. They click the **Delete** button. When clicked, the deleteDoc() JavaScript function is called. If the user elects to send an e-mail notification, the WebInvitation form is returned to the user with special parameters added to the Query String. The user completes the form and addresses the form as appropriate and then clicks the **Send** button. This submits the document/page to the Domino server where the e-mail is sent. When finished, the user is returned to the appropriate page.

11.3.3 Implementing the e-mail notification feature

Implementing the email notification feature involves inserting new design elements and modifying existing, out-of-the-box ones. It is important to note that each time you upgrade your Sametime server you will need to re-apply these changes. Our experience has been that it takes less than one day for a skilled developer to re-apply the changes and test them.

Tip: When making design changes to the Meeting Center template or any other Sametime template, it is a good practice to make a few changes and then test them. The templates and databases can be a little sensitive.

Here are the high-level steps to implementing these changes:

1. Obtain a replica-copy of the Sametime Online Meeting Center.
2. If you are doing this after an upgrade, evaluate the differences between the existing template and the new template.
3. Copy the following design elements from the sample database file included with this book, STMtgCtr-Notification.nsf:
 - CustomEmailCheck (subform)
 - WebInvitation (form)
 - WebInvitation-x (multiple subforms)
 - WebQuerySaveInvitation (agent)
4. Manually modify the following existing design elements using the Notes Designer client as described later in this chapter:
 - WebConference - Form
 - WebValidation-IE,-Moz,-N4 - Subform
 - WebRunRestrictApplet-N4 - Subform
 - WebConfirmDelete - Subform
 - (SametimeWebQuerySave) - Agent
 - \$\$ViewTemplate for vwCalendar
 - \$\$ViewTemplate for vwTodayConf
 - \$\$ViewTemplate for vwWebActiveMeetings
 - \$\$ViewTemplate for vwWebAdminHiddenMeetings
 - \$\$ViewTemplate for vwWebAllMeetings
 - \$\$ViewTemplate for vwWebFinishedMeetings
 - \$\$ViewTemplate for vwMyMeetings
 - \$\$ViewTemplate for vwRecordedMeetings
 - \$\$ViewTemplate for vwWebScheduledMeetings
 - \$\$ViewTemplate for vwWebTodaysMeetings
 - \$\$ViewTemplate for (\$WebSearch)
5. Refresh the design of the Sametime Online Meeting Center (stconf.nsf) database from the new, customized template on a non-production server.
6. Test the changes.
7. Once your changes have been verified in a non-production environment, deploy the changes into your production environment.

Note: These high-level steps are just guidelines. They are by no means a definitive process.

11.3.4 Design changes

In this section, we list all the design elements that we either created or modified to construct the e-mail notification feature and how to apply the design changes to the template. This section is somewhat ordered in the way that design elements are used within the feature.

Note: Manual design changes to the template are not as simple as cut and paste. You should pay careful attention when modifying any pass-thru HTML or JavaScript. Finding syntax error can be tedious and time consuming.

Form: WebConference

The WebConference form is used for creating, editing, viewing, and attending meetings. Changes made to this form allow us to store whether or not an e-mail notification has been sent and if so to whom. The form already exists in the template, so these changes need to be applied manually.

To apply the changes, you will need to copy several fields and computed text lines from the sample template into your template. Here are the basic steps to follow:

1. Open the WebConference form in the sample template using Notes Designer.
2. Copy all the fields and computed text in the form and then close the form.
3. Open the WebConference form in the Sametime Online Meeting Center template (stconf31.ntf) and locate the field named RedirectToPrefix. Paste the fields and computed text immediately after the field.
4. Save and close the form.

Table 11-1 lists all the fields that were added to the form and their purpose.

Table 11-1 WebConference form fields

Name	Comments
SendInvitation	Used to flag whether or not we want to send an invitation.
Invitation_Sent	Set to 1 if an invitation has been sent, otherwise 0.

Name	Comments
SendInvitation_Update	When set tells the SametimeWebQuerySave agent whether or not the Notification request is an update to an existing meeting.
Invitation_ForceSend	Temporary field used to skip the “Do you want to send...” prompt. Used when the password needs to be reset.
Invitation_Saved_Message	Store the text message from the invitation that was last sent.
Invitation_Saved_List	The list of users that the last e-mail notification was sent to.
Invitation_Saved_cn	Common names of the last users the notification was sent to.
Invitation_Saved_ug	Flag stating whether or not the recipient is a group or user.
Invitation_Parent	UNID of parent.
dsplInvitation_Saved_List	Used when generating HTML pass thru.
dsplInvitation_Saved_ug	Used when generating HTML pass thru.
dsplInvitation_Saved_cn	Used when generating HTML pass thru.

Subform: CustomEmailCheck

The CustomEmailCheck subform queries the user to see if they would like to send an e-mail notification or not. It takes into account whether or not an e-mail notification has been sent before.

This subform was added, so you simply need to copy it from the sample template into the new template.

Subform: WebValidation-x

WebValidation-x represents three subforms: WebValidation-IE, WebValidation-Moz, and WebValidation-N4. Each subform is designed for a specific browser and is inserted into the WebConference form dynamically using a computed-subform.

Each of these subforms contains a JavaScript function named checkForm(). This function is called when the **Save** button is clicked on the WebConference form. It performs some form validation and then, if appropriate, submits the page to the

server. It is in this function that we insert the CustomEmailCheck subform so we can query the user in regards to the notification.

To apply these changes, you need to find the correct location in each subform and insert the CustomEmailCheck subform. The following are the basic steps to follow for each of the three subforms:

1. Open the new template using the Notes Designer client and navigate to the subforms section.
2. Open the subform and find the checkForm() JavaScript function. Then insert the subform as outlined in Example 11-2.
3. Save and close the subform.

Example 11-2 Where to insert in the WebValidation-x subforms

```
repeatEditType.value = "1";  
repeatEditDate.value = getStartDateString();  
  
>>> Insert CustomEmailCheck subform here <<<  
  
meetingSubmitted = 1;  
submit();
```

Subform: WebRunRestrictApplet-N4

The WebInvitation form, which is the form used to send the e-mail notification, uses the Sametime Directory applet to allow users to select users and groups from the directory. When the user is using the Netscape 4.x, the applet is displayed using the WebRunRestrictApplet-N4 subform. The JavaScript in the subform is somewhat hardwired to the WebConference form, so we needed to make changes in order for the applet to work with both forms.

To apply these changes, manually insert the code listed in bold in Example 11-3 into the subform.

Example 11-3 WebRunRestrictApplet-N4 code changes

```
<script language="JavaScript">  
<!--  
// Customization  
var frmVInvite;  
// get reference to determine if parent is WebConference or Invitation form  
var frm0 = window.opener.document.forms[0];  
var isVInvite = false;  
// check if Invitation, then set appropriate flags  
if (frm0.name == '_frmInvitation') {  
    isVInvite = true;      // yes – Invitation form  
    frmVInvite = frm0;  
}
```

```

} else {
    isVInvite = false;    // no - WebConference
    frmVInvite = null;
}

function runGetCommonNames()
{
    if (isVInvite) {
        cnNames = frmVInvite.invitees_cn.value;
    } else {
        cnNames =
window.opener.document.layers["tab3"].document.frm.invitees_cn.value;
    }
    return cnNames;
}

function runGetFullNames()
{
    if (isVInvite) {
        fullNames = frmVInvite.InviteeList.value;
    } else {
        fullNames = window.opener.document._frmConference.InviteeList.value;
    }
    return fullNames;
}

function runGetNameTypes()
{
    if (isVInvite) {
        nameTypes = frmVInvite.invitees_ug.value;
    } else {
        nameTypes = window.opener.document._frmConference.invitees_ug.value;
    }
    return nameTypes;
}

function closeDirectoryApplet(okVal)
{
    if (okVal == 1)
    {
        if (isVInvite) {
            frmVInvite.invitees_cn.value =
document.RestrictApplet.getCommonNames();
            frmVInvite.InviteeList.value =
document.RestrictApplet.getDistinguishedNames();
            frmVInvite.invitees_ug.value =
document.RestrictApplet.getUserGroupFlags();
        }
    }
}

```

```

        else
        {
            window.opener.document.layers["tab3"].document.frm.invitees_cn.value
= document.RestrictApplet.getCommonNames();
            window.opener.document._frmConference.InviteeList.value =
document.RestrictApplet.getDistinguishedNames();
            window.opener.document._frmConference.invitees_ug.value =
document.RestrictApplet.getUserGroupFlags();
        }

        self.close();
    }
    else
    {
        self.close();
    }
}
// -->
</script>

```

Subform: WebConfirmDelete

The WebConfirmDelete subform contains the JavaScript code that is triggered when the user clicks the Delete button on the WebConference form when a document is read/view mode. The customizations determine whether or not a notification has previously been sent and then if so they query the user on whether or not they would like to send another notification stating that the meeting has been cancelled.

To apply these changes, manually modify the code using following guidelines.

1. Open the sample template using the Notes Designer client.
2. Open the WebConfirmDelete subform.
3. Locate all the sections highlighted in bold in Example 11-4 on page 319. These areas represent the changes you need to make to the out-of-the-box code.
4. Open the new template using the Notes Designer client.
5. Open the WebConfirmDelete subform.
6. Copy the appropriate sections of code from the WebConfirmDelete subform in the sample database into the new template.
7. When finished with all the changes, save and close.

Important: The modifications printed in Example 11-4 contain computed text fields that cannot be displayed in this document. Make sure to copy changes from the Sample database.

Example 11-4 deleteDoc() JavaScript function

```
<script language="JavaScript">
<!--
var wRepDel = null;
STR_DELMSG = "";
STR_MEETING_CANCELLED = "This meeting has not taken place yet.\n\nDo you want
to send an e-mail notification?\n\n";
STR_MEETING_CANCELLED += 'Click \"OK\" to save and send an e-mail
notification.\n\n';
STR_MEETING_CANCELLED += 'Otherwise, click \"Cancel\" to save and continue.';

function deleteDoc()
{
    // Launch the agent without the login parameter if the user is Anonymous
    abURL = '/<computedtext>/WebDelete?OpenAgent<computedtext>';
    // Launch the agent with the login parameter if the user has
    authenticated
    abURL = '/<computedtext>/WebDelete?OpenAgent<computedtext>&login';
    if (confirm(STR_DELMSG))
    {
        // Customization: if the meeting has been cancelled a notification
        should be sent
        // but only if the meeting was scheduled in the future
        var now = new Date();
        var scheduleTime = new Date(<computedtext>);
        var scheduleDateTime = new Date(scheduleTime.getTime() + zonediff);
        var diff = scheduleDateTime.getTime() - now.getTime();
        // now get the state of the meeting
        var state = <computedtext>;
        if (diff > 0 && state == 0 && <computedtext> == 1) {
            if (confirm(STR_MEETING_CANCELLED)) {
                abURL =
                '/<computedtext>/frmInvitation?OpenForm&parentunid=<computedtext>&login&delete=
                1';
            }
        }
        location.replace(abURL);
    }
}

function deleteRepeatDoc()
{
    // Launch the agent without the login parameter if the user is Anonymous
```

```

        abURL =
'</computedtext>/WebRepeatDelete?OpenForm<computedtext>&i sent=<computedtext>&i
parent=<computedtext>';
        // Launch the agent with the login parameter if the user has
authenticated
        abURL =
'</computedtext>/WebRepeatDelete?OpenForm<computedtext>&login&i sent=<computedt
ext>&i parent=<computedtext>';
        wheight="275";
        wwidth="525";
        wleft="200";
        wtop="200";
        opts='resizable=1,width=' + wwidth + ',height='+wheight+
',status=yes,left=' + wleft + ',top='+wtop;
        if ((wRepDel != null) && (!wRepDel.closed))
            wRepDel.focus();
        else
            wRepDel=window.open(abURL,'abRepeatDeleteWindow',opts);
    }
function editDoc()
{
    var uniqueifier = new Date().getTime().toString();
    // Launch the agent without the login parameter if the user is Anonymous
    abURL = '/'
/meeting/
?editdocument&' + uniqueifier;
    // Launch the agent with the login parameter if the user has
authenticated
    abURL = '/'
/meeting/
?editdocument&' + uniqueifier + '&login';
    location.href = abURL;
}
function boxClose()
{
    if (wRepDel != null)
    {
        if (!wRepDel.closed)
            wRepDel.close();
    }
}
// -->
</script>

```

Form: WebRepeatDelete

The WebRepeatDelete form is a dialog box that is displayed when a user is deleting a meeting that repeats. The dialog box allows the user to select which

meetings they would like to delete. The customizations enable the user to send an e-mail notification before deleting the meeting.

Changes to this form are a mixture of adding fields, JavaScript, and computed text. Apply the changes following these steps:

1. Open the sample template using the Notes Designer client.
2. Open the WebRepeatDelete form.
3. Copy the following fields: Invitation_Sent and Invitation_Parent
4. Open the new template using the Notes Designer client.
5. Open the WebRepeatDelete form.
6. Paste the two fields from step 3.
7. Insert the following text immediately before the checkForm() JavaScript function:

```
STR_MEETING_CANCELLED = "This meeting has not taken place yet.\n\nDo you want to send an e-mail notification?";
STR_MEETING_CANCELLED += 'Click \"OK\" to save and send an e-mail notification.\n\n';
STR_MEETING_CANCELLED += 'Otherwise, click \"Cancel\" to save and continue.';
```

8. Copy the highlighted sections of code from the sample template and insert it into the new template in the exact same location (see Example 11-5).

Example 11-5 WebRepeatDelete JavaScript modifications

```
// Launch the agent without the login parameter if the user is Anonymous
rdURL = '/<computed text>/WebDelete?OpenAgent&' + delString;
// Launch the agent with the login parameter if the user has
authenticated
rdURL = '/<computed text>/WebDelete?OpenAgent&' + delString + '&login';

var V_EMAIL_CONFIRM_TEXT = '';

if ( <computed text> == '1') {
    V_EMAIL_CONFIRM_TEXT = 'An e-mail notification has been sent in the
past.\n\nDo you want to send out an updated e-mail notification?\n\n';
} else {
    V_EMAIL_CONFIRM_TEXT = 'Do you want to send out an e-mail
notification now?\n\n';
}

V_EMAIL_CONFIRM_TEXT += 'Click \"OK\" to save and send an e-mail
notification.\n\n';
V_EMAIL_CONFIRM_TEXT += 'Otherwise, click \"Cancel\" to save and
continue.';

if (confirm(V_EMAIL_CONFIRM_TEXT)) {
```

```

        rdURL = '/<computed
text>/frmInvitation?OpenForm&parentunid=&delete=1' + delString +
'&meetingid=<computed text>' + '&login';
    }

// * end - customization *

    window.opener.location.replace(rdURL);
    self.close();

```

Agent: (SametimeWebQuerySave)

The (SametimeWebQuerySave) is triggered whenever the WebConference form is submitted or save. The customizations in this agent determine whether or not the user has elected to send the notification or not. If so, then the frmInvitation form is returned to the user.

To apply these changes, you will need to manually modify the existing agent in the template as follows:

1. Locate the section shown in Example 11-6 in the agent and insert the code listed in bold.

Example 11-6 (SametimeWebQuerySave) modifications

```

'Variables used when computing the end date/time from duration
    Dim dString As String
    Dim hourVal As Integer
    Dim dayVal As Integer
    Dim minVal As Integer

    ' * start - customization *
    Dim itemVSendInvitation As NotesItem
    ' * end - customization *

    Dim pathreldb As String

```

2. Locate the section shown in Example 11-7 in the agent and insert the code listed in bold.

Example 11-7 (SametimeWebQuerySave) modifications

```

InstantiateObjects
deldbpath = getdbpath
pathinfo = delnote.path_info(0)

Set note = session.DocumentContext
' *** customization - reset field ***
note.Invitation_ForceSend = "0"

```

```
pathreldb = note.RedirectToPrefix(0)
```

3. Locate the section shown in Example 11-8 and insert the code listed in bold.

Example 11-8 (SametimeWebQuerySave) modifications

```
If note.HasItem( "RepeatIds" ) Then
    'the meeting id for the first of the repeat set is gathered in the
RepeatSave script
    'we use that value below and then remove the item from the main document
    idVal = note.RepeatIds(0)
    Call note.RemoveItem("RepMeetIDs")
Elseif note.HasItem("STRepeatEditUNID") Then
    idVal = note.STRepeatEditUNID(0)
Else
    idVal = note.universalid
End If
Print |Content-Type: text/html|
' *** start: customization ***
If (ShouldReturnInvitationForm(note) = False) Then
' *** end: customization ***
    If note.User_Name(0) = "Anonymous" Then
        Print |Location: | & pathreldb & |meeting/| & Cstr(idVal) &
|?opendocument|
    Else
        Print |Location: | & pathreldb & |meeting/| & Cstr(idVal) &
|?opendocument&login|
    End If
        ' *** start: customization ***
End If
        ' *** end: customization ***

End If
' * start -customization *
If (ShouldReturnInvitationForm(note)) Then
    note.SendInvitation = ""
    Goto SendInvitationForm
End If
' * end - customization *
```

4. Locate the section in Example 11-9 in the agent and insert the code listed in bold.

Example 11-9 (SametimeWebQuerySave) modifications

```
ErrorRoutine:
    Print "<br>" & Error & " Line: " & Cstr(Erl) & " (SametimeWebQuerySave)"
    Goto EndSub
```

SendInvitationForm:

```

    Print |Location: | & pathreldb &
|frmInvitation?OpenForm&Login&ParentUNID=| & note.universalid & |&update=| &
note.SendInvitation_Update(0)
    'Print |</SCRIPT>|
    Goto EndSub

```

5. Add the subroutine shown in Example 11-10 to the agent.

Example 11-10 (SametimeWebQuerySave) modifications

```

Function ShouldReturnInvitationForm(note As NotesDocument) As Integer
    Dim rc As Integer
    rc = False

    Set itemVSendInvitation = note.getFirstItem("SendInvitation")
    If Not (itemVSendInvitation Is Nothing) Then
        If (itemVSendInvitation.text = "1") Then
            rc = True
        End If
    End If

    ShouldReturnInvitationForm = rc
End Function

```

Form: WebInvitation

A better name for this form would be WebNotification, but alas, it is not. This form serves as the template for the e-mail notification. The form contains the following input fields: send to, subject, and message. The form leverages the Sametime Directory applet, which is also used to change the moderator and restrict the meeting.

Note: The SaveOptions field is set to “0” to prevent the document from being saved.

To apply this form, copy the form from the sample template into the new template.

Subform: WebInvitation-x

WebInvitation-x represents five subforms: WebInvitation-IE, WebInvitation-N4, WebInvitation-Moz, WebInvitation-Old, and WebInvitation-UN. Each of these subforms is browser specific. The subforms are used in the WebInvitation form to insert the SendTo field and to properly integrate the Sametime Directory applet.

To apply these subforms, copy them from the sample template into the new template.

Agent: WebQuerySaveInvitation

This agent is triggered when the WebInvitation form is submitted. The agent is responsible for sending the e-mail notification to anyone listed in the SendTo and CopyTo fields and also for updating the meeting document with the people that were notified. This allows the WebInvitation form to prepopulate the SendTo field if the user were to change or cancel the meeting.

To apply this agent, copy the agent from the sample template into the new template.

11.3.5 Validating the changes

Table 11-2 through Table 11-7 on page 330 show a quick set of tests you can run to validate that the changes were properly applied.

Table 11-2 Test case 1

Item	Description
Description	New meeting, no restrict list, and send notification.
Input(s)	None.
Expected Outcome	E-mail notification is sent to users in the Send To field of the Notification form.
Steps to reproduce	<ol style="list-style-type: none">1. Click on the New Meeting link and the Meeting (WebConference) form is displayed.2. Fill in the details, but do not restrict the meeting.3. Click the Save button. You will be prompted whether or not you want to send a notification.4. Click OK. The WebInvitation (Notification) form is displayed.5. Select individuals or groups to notify and click the Send button.6. An e-mail is sent to the participants entered in SendTo field in the WebInvitation form.

Table 11-3 Test case 2

Item	Description
Description	New meeting, individuals, or groups selected in the restrict to field, and send notification.
Input(s)	None.
Expected Outcome	E-mail notification is sent to users in the restrict to list of the meeting document and those additional individuals entered in the SendTo field in the Notification form.
Steps to reproduce	<ol style="list-style-type: none"> 1. Click the New Meeting link. A Meeting (WebConference) form is displayed. 2. Enter meeting details and restrict the meeting to a certain individuals or groups. 3. Click the Save button. You will be prompted whether or not you want to send a notification. 4. Click OK. The Notification form is displayed with the SendTo field populated with the individuals or groups in the meeting document restrict to list. 5. Select other individuals or groups to be notified, but only those in the RestrictTo field will be saved for future use. Click the Send button. 6. An e-mail is sent to the participants entered in SendTo field in the WebInvitation form.

Table 11-4 Test case 3

Item	Description
Description	Edit/Update meeting, no restrict list, and send notification.
Input(s)	Meeting document from Table 11-2.
Expected Outcome	E-mail notification is sent to users in the Send To field of the Notification form.
Steps to reproduce	<ol style="list-style-type: none"> 1. Open the Scheduled meetings view in the Online Meeting Center and find the document create in Test Case 1. 2. Open the document and click the Edit button. 3. Optionally, update the meeting details, and do not restrict the meeting. 4. Click the Save button. You will prompted whether or not you want to send a notification. 5. Click OK. The notification form is displayed and the users and/or groups who were notified in Test Case 1 are pre-populated and displayed in the Send To box. (Comments will also be pre-populated.) 6. Optionally, add or remove individuals or groups to notify. Optionally, enter comments. 7. Click the Send button. 8. An e-mail will be sent to those individuals and/or groups in the Send To field.

Table 11-5 Test case 4

Item	Description
Description	Edit/Update meeting, individuals, or groups selected in the Restrict To field, and send notification.
Input(s)	Meeting document created in test case 2 in Table 11-3.
Expected Outcome	E-mail notification is sent to users in the restrict to list of the meeting document and those additional individuals entered in the SendTo field in the Notification form.
Steps to reproduce	<ol style="list-style-type: none"> 1. Open the Scheduled meetings view in the Online Meeting Center and find the document created in Test Case 2. 2. Open the document and click on the Edit button. 3. Optionally, update the meeting details. Optionally, add additional individuals or groups to the restrict list. 4. Click the Save button. You will prompted whether or not you want to send a notification. 5. Click OK. The notification form is displayed and the users and/or groups who are in the Restrict To list are pre-populated in the Send To box. (Comments will also be pre-populated.) 6. Optionally, add or remove individuals or groups to notify. Optionally, enter comments. 7. Click the Send button. <p>An e-mail will be sent to those individuals and/or groups in the Send To field.</p>

Table 11-6 Test case 5

Item	Description
Description	Delete Meeting, no restrict to list, and a notification was sent earlier.
Input(s)	Meeting document updated in test case 3.
Expected Outcome	E-mail notification is sent to users in the restrict to list of the meeting document and those additional individuals entered in the SendTo field in the Notification form.
Steps to reproduce	<ol style="list-style-type: none"> 1. Open the Scheduled meetings view in the Online Meeting Center and find the document updated in Test Case 3. 2. Open the document. 3. Click the Delete button. You will be prompted whether or not they want to delete the document. Click OK. 4. You will be prompted again asking whether or not you want to send a notification. Click OK. 5. The Notification form is displayed and the SendTo, Comment, and Subject fields are pre-populated. Click Send. 6. An e-mail is sent to those people in the SendTo field on the Notification form.

Table 11-7 Test case 6

Item	Description
Description	Delete meeting, and no meeting notification was sent previously.
Input(s)	None.
Expected Outcome	Meeting document gets deleted.
Steps to reproduce	<ol style="list-style-type: none"> 1. Create a new meeting document. 2. Click the Save button, but do not send a notification. The meeting document is displayed in view/read mode. 3. Click the Delete button. 4. You will not be prompted to send a notification. No notification will be sent.

Note: Test cases should also be performed on repeating meetings.

11.4 Summary

In the chapter, we have discussed how to customize the look and feel of the Sametime Online Meeting Center template and how to add a feature that sends an e-mail notification whenever a meeting is created, updated, or deleted.



Ideas for customization and integration

This chapter will explain just a few of the things that are possible with Lotus Sametime when you start to broaden your horizons past the idea of traditional instant messaging. Although the Sametime Connect client is not customizable, with the power of the Sametime Toolkits you can build very powerful systems that can interact with each other through the instant messaging infrastructure.

In this chapter, we hope to help you realize some of the great things that are possible, and help open up your imagination to the hidden value that you have within your instant messaging infrastructure.

12.1 Why customize and integrate?

The IBM Lotus Sametime Connect client at this time is not available for customization, unless you are privileged to have access to the necessary source code. As you can see from the examples within this chapter and throughout this redbook however, Sametime is capable of much more than simply sending text instant messages. In the examples within this chapter, we will show you how to send data in XML format, which can in turn represent graphic images directly within a customized Sametime Java Connect client.

Customization allows users and organizations to further leverage the capabilities of Sametime beyond its traditional usage as an instant messaging client, or as an e-meeting hosting tool. Fortunately, Lotus does provide the Sametime SDK, which contains the different toolkits. This allows for customization and integration with other applications on a variety of platforms.

12.2 Using Sametime to send data

You should, by now, be very familiar with Sametime's ability to send text messages between clients. This is the foundation for Sametime's instant messaging functionality. But the Instant Messaging Service of the Client Toolkit also allows binary data to be sent between clients. The Instant Messaging Service provides the `sendData()` method to send binary data to another party. The `dataReceived()` method is called in response by the receiving party.

In this section, we describe a Sametime Bot that sends information about a customer's orders to a Java applet client. An end user enters a customer order number into the applet. The applet sends the information to the bot using a standard text message. The bot receives the message and looks up the order details. It sends a text message back to the applet, echoing the message it received. But it also sends back binary data containing full details about the requested order. The applet receives this data and updates its display so the end user can see the full order summary.

This example is made up of two parts: a `RichTextBot` Java application and a `RichTextClient` Java applet. Both are developed using the Java Client Toolkit.

12.2.1 The RichTextBot Sametime Bot

The `RichTextBot` is a Java application very similar to the one described in 3.3.1, "The Echo Bot" on page 56. It logs in to the Sametime server and waits to be contacted by the client applet. When it receives a text message from the applet,

the `textReceived()` method is called. The bot retrieves the order number sent to it using the `getText()` method of the `ImEvent` object:

```
public void textReceived(ImEvent e)
{
    String messageText = e.getText();
    int sprInt = new Integer(messageText).intValue();
```

Example 12-1 shows how the bot uses a Java switch / case statement to select the details for a particular order. It uses the `sendData()` method of the `Im` object to send four lots of binary data back to the applet. The bot then uses the `sendText()` method to echo back to the applet the order number it sent.

Example 12-1 Portion of bot's textReceived() method

```
switch (sprInt)
{
    case 1:
        strDescription = "Arizona sunset watercolour";
        e.getIm().sendData(true, 100001, 5, strDescription.getBytes());
        strReportedBy = "Dave";
        e.getIm().sendData(true, 100001, 6, strReportedBy.getBytes());
        strStatus = "Awaiting delivery to customer";
        e.getIm().sendData(true, 100001, 7, strStatus.getBytes());
        strImageURL = "AZSunc.jpg";
        e.getIm().sendData(true, 100001, 8, strImageURL.getBytes());
        e.getIm().sendText(true, "SPR " + messageText);
        break;

    case 2:
        strDescription = "The Diver's watch";
        e.getIm().sendData(true, 100001, 5, strDescription.getBytes());
        strReportedBy = "Max";
        e.getIm().sendData(true, 100001, 6, strReportedBy.getBytes());
        strStatus = "Out of oxygen";
        e.getIm().sendData(true, 100001, 7, strStatus.getBytes());
        strImageURL = "watchlg.jpg";
        e.getIm().sendData(true, 100001, 8, strImageURL.getBytes());
        e.getIm().sendText(true, "SPR " + messageText);
        break;

    case 3:
        strDescription = "A wearable PC";
        e.getIm().sendData(true, 100001, 5, strDescription.getBytes());
        strReportedBy = "Charlie";
        e.getIm().sendData(true, 100001, 6, strReportedBy.getBytes());
        strStatus = "Waiting for taxi";
        e.getIm().sendData(true, 100001, 7, strStatus.getBytes());
        strImageURL = "wearpclg.gif";
```

```

        e.getIm().sendData(true, 100001, 8, strImageURL.getBytes());
        e.getIm().sendText(true, "SPR " + messageText);
        break;
    }
}

```

The `sendData()` method takes four parameters:

- ▶ A boolean, indicating whether or not the text should be encrypted.
- ▶ An int, specifying the data type. The values 0 - 100,000 are reserved for internal Sametime/Lotus/IBM use, so we chose the value 100001.
- ▶ An int, specifying the data subtype. We chose an incremental number for each one of the four pieces of data the bot sends back.
- ▶ An array of bytes, representing the actual data. The bot uses the `getBytes()` method of the `String` class, which conveniently returns a byte array.

If a normal Connect client user were to contact the bot, they would only see the customer order number echoed back to them. The Connect client only displays the information sent by the `sendText()` method. This is received in the `textReceived()` method. As Figure 12-1 shows, the information sent by the four calls to the `sendData()` method is not displayed to a Connect user.



Figure 12-1 Connect client only sees information sent using `sendText()`

We now move on to look at the `RichTextClient` Java applet, and how it displays the extra information sent by the `RichTextBot`.

12.2.2 The RichTextClient Sametime applet

The RichTextClient applet provides the user interface for this example. Figure 12-2 shows what it looks like when launched in JBuilder's test environment.

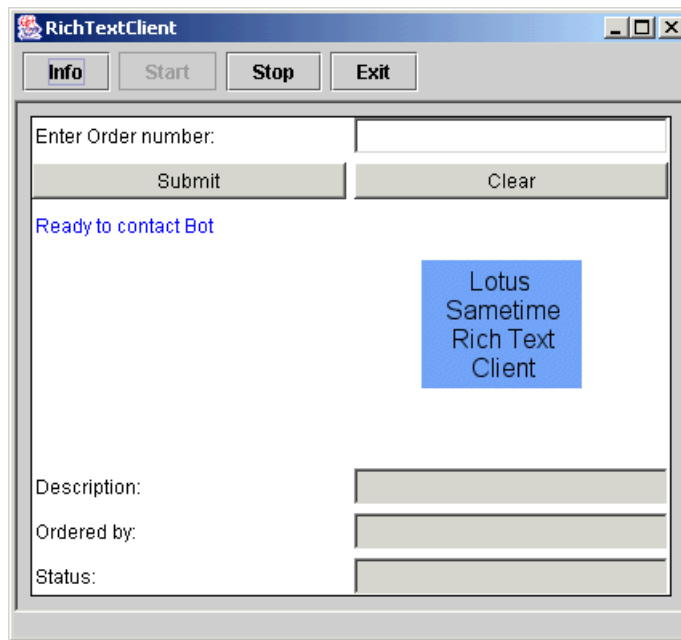


Figure 12-2 The RichTextClient applet

The applet contains a text field for the end user to enter an order number. There are also buttons to submit the order number to the RichTextBot application, and to clear the screen's details. There is a small text panel where status information is displayed. The main portion of the applet is made up of an area to display an image of the product ordered, and three text fields to show information about the order. These text fields are read-only.

When the applet is loaded, its `init()` method is called. This is where all the user interface initialization is performed. It is also where the applet logs in to the Sametime server. Once it has logged in, the `loggedIn()` method is called and the applet uses a Resolver object to get a handle on the RichTextBot.

Example 12-2 RichTextClient's loggedIn() method

```
public void loggedIn(LoginEvent e)
{
    announce("loggedIn");
    m_resolver = m_lookupService.createResolver(false, // Return all matches
                                                false, // Non-exhaustive
lookup
                                                true, // Return resolved
users
                                                false); // Do not return
resolved groups

    m_imService =
(InstantMessagingService)m_session.getCompApi(InstantMessagingService.COMP_NAME
);
    m_imService.registerImType(ImTypes.IM_TYPE_CHAT);
    m_imService.addImServiceListener(this);

    m_resolver.addResolveListener(this);
    m_resolver.resolve(m_strUserName);
}
```

If the RichTextBot is resolved correctly, the resolved() method is called. Here, the resolved STUser object is assigned to the member variable richTextBot, and then the two buttons are made active. An instant messaging session between the applet and the RichTextBot is then created.

Example 12-3 RichTextClient's resolved() method

```
public void resolved(ResolveEvent event)
{
    if (event.getName().compareTo(m_strUserName) == 0)
    {
        announce("Ready to contact Bot");
        richTextBot = (STUser) event.getResolved();
        btnSubmit.setEnabled(true);
        btnClear.setEnabled(true);
        im = m_imService.createIm(richTextBot, EncLevel.ENC_LEVEL_NONE,
ImTypes.IM_TYPE_CHAT);
    }
}
```

The applet is then ready to accept an order number and query the RichTextBot for the order's full details. Figure 12-3 on page 339 shows the result of entering an order number and clicking the **Submit** button.

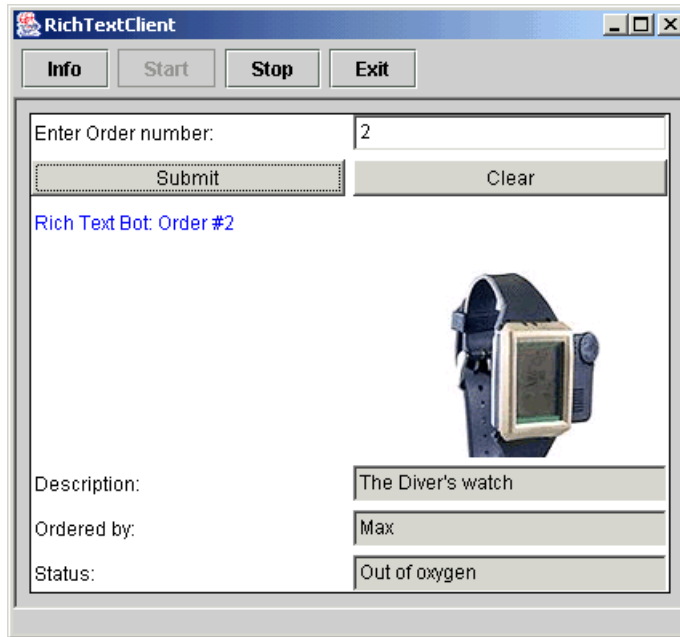


Figure 12-3 Applet receiving back order information from RichTextBot

Clicking the **Submit** button calls the applet's `requestData()` method. The order number is retrieved from the input field and sent to the bot using the `sendText()` method of the `Im` object.

Example 12-4 RichTextClient's `requestData()` method

```
void requestData()
{
    boolean imExist = false;
    Im currentIm = null;
    for(int i = 0; i < m_ImOpened.size(); i++)
    {
        currentIm = (Im) m_ImOpened.elementAt(i);
        if (currentIm.equals(im))
        {
            imExist = true;
            im = currentIm;
            sendText(im);
            break;
        }
    }

    if (!imExist)
    {
```

```

        m_ImOpened.addElement(im);
        im.addImListener(this);
        im.open();
    }
}

public void sendText(Im im)
{
    sprNumber = tfSPR.getText();
    im.sendText(false, sprNumber);
}

public void imOpened(ImEvent e)
{
    sprNumber = tfSPR.getText();
    e.getIm().sendText(false, sprNumber);
}

```

As discussed in the previous section, the RichTextBot receives the customer order number in its `textReceived()` method. It then uses its `sendText()` and `sendData()` methods to send back full information about the order.

The applet receives this information in its `textReceived()` and `dataReceived()` methods. In the `textReceived()` method, the applet writes out the text sent by the bot to the status panel.

Example 12-5 Applet's `textReceived()` and `announce()` methods

```

public void textReceived(ImEvent e)
{
    String receivedText = e.getText();
    String receivedFrom = e.getIm().getPartner().getDisplayName();
    announce(receivedFrom + ": " + receivedText);
}

void announce(String msg)
{
    System.out.println(msg);
    statusBar.setText(msg);
}

```

In the `dataReceived()` method (Example 12-6 on page 341), the applet uses the `getData()` method of the `ImEvent` object to get the data sent by the bot. It turns this byte array into a `String` object. The applet retrieves the data subtype sent by the bot using the `getDataSubType()` method of the `ImEvent` object. It then uses a `switch / case` statement to update the correct text field with the new information.

Example 12-6 Applet's `dataReceived()` method

```
public void dataReceived(ImEvent e)
{
    announce("dataReceived");
    byte[] myBytes = e.getData();
    String myString = new String(myBytes);
    int dataSubType = e.getDataSubType();
    switch (dataSubType)
    {
        case 5: tfDescription.setText(myString); break;

        case 6: tfReportedBy.setText(myString); break;

        case 7: tfStatus.setText(myString); break;

        case 8: itemUrlChanged(myString); break;
    }
}
```

Note: At the time of the writing of this redbook, we originally intended to use the COM Toolkit to build a Visual Basic client for this example, rather than a Java applet. Unfortunately, a bug was discovered in the COM Toolkit's implementation of the `dataReceived()` method. The method is correctly called when data is received, and the data type and subtype can be retrieved successfully. However, the data content is always reported as being empty. This issue has been reported to the Sametime development team and is expected to be addressed within a future release

Please refer to the IBM Lotus Software support site for any fixes or updates to this issue at:

<http://www-3.ibm.com/software/lotus/support/sametime/support.html>

12.3 Alternative approaches to Single Sign-On (SSO)

Sametime supports community security through the authentication of community members. Authentication is the process of verifying that a user is who she claims to be. The Community Service is the core Sametime service you must use in order to authenticate and login to the Sametime server. The Community Service provides two methods to authenticate with the server:

- ▶ `loginByPassword()`: Enables login using a login name and password.
- ▶ `loginByToken()`: Enables login using a login name and a temporary login token.

Logging in with a password may be a workable solution in some circumstances. However, it may not be possible or allowed in other circumstances. As an alternative, the Sametime server includes two separate security features capable of generating an authentication token:

- ▶ WebSphere / Domino Single Sign-On (SSO) authentication: This authentication method uses Lightweight Third Party Authentication (LTPA) tokens. This method was new in Sametime 3.0.
- ▶ Secrets and Tokens authentication databases: All previous releases of Sametime used only the Secrets and Tokens authentication databases to create authentication tokens. The Sametime 3.1 server supports both the WebSphere/Domino SSO feature and the Secrets and Tokens authentication system.

The recommended authentication mode for applications that work with a Sametime 3.1 server is LTPA. However, if the Domino server uses a DSAPI filter for authentication instead of LTPA, the application can authenticate with a Secrets and Tokens token.

The *Sametime 3.1 Java Toolkit Developer's Guide* details how a Domino application can generate a Sametime token. This allows a user who has logged into the Domino application to log in to the Sametime server without being challenged to authenticate again.

However, you may wish to integrate Sametime into an environment that does not contain WebSphere or Domino. In this case, you will not have access to an LTPA token, so you cannot use that method. You could use Java to open an HTTP connection to a Sametime-enabled Domino application and generate a token that way, but performance and scalability may become a problem. You may be able to implement a solution at the architecture level by creating a DSAPI filter.

In the next section, we discuss how you can use the Community Server Toolkit to generate a token for a Sametime user.

12.3.1 The TokenGenerator servlet

The Community Server Toolkit contains the SATokenService component, which can generate Sametime tokens for valid users. The TokenGenerator is a server application built as a Java servlet that takes a user name as a parameter. It logs into the Sametime server as a server application and generates a token for that user. This token can then be used to log in to the server with the `loginByToken()` method of the Client Toolkit's CommunityService component.

The TokenGenerator imports a number of packages (Example 12-7), including the com.lotus.sametime.token package that contains the SATokenService component.

Example 12-7 Package imports for TokenGenerator servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

import com.lotus.sametime.community.*;
import com.lotus.sametime.core.comparch.*;
import com.lotus.sametime.core.types.*;
import com.lotus.sametime.core.util.connection.*;
import com.lotus.sametime.lookup.*;
import com.lotus.sametime.token.*;
```

The TokenGenerator is a servlet, so it extends the javax.servlet.http.HttpServlet class. The servlet's doGet() method is called whenever an HTTP GET request is made for it, that is, whenever it's URL is called from a browser. The doGet() method in turn calls the getToken() method, passing in the user name as a parameter. Depending upon what the getToken() method returns, either the token itself or an error message is printed to the browser. It is at this point that you would add logic to incorporate the token into your SSO scenario. You could, for example, include the token as a parameter in an <applet> tag on an HTML page.

Example 12-8 TokenGenerator doGet() method

```
/**Process HTTP requests */
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();

    // Extract user name parameter here - hard-coded in this example...
    String strUserName = "CN=John Barrow,0=itsportal";

    if (strUserName == null && strUserName.length() == 0)
    {
        out.println("<html><head></head><body>");
        out.println("<h1>There has been an error - no username supplied</h1>");
        out.println("</body>");
        out.println("</html>");
        System.out.println("Failed validation");
        return;
    }
}
```

```

else
{
    System.out.println("Passed validation");
}

String strToken = getToken(strUserName);

if (strToken.compareTo("Error") == 0)
{
    out.println("<html><head></head><body>");
    out.println("<h1>Unable to generate Sametime Token for current
user.</h1>");
    out.println("</body>");
    out.println("</html>");
}
else
{
    out.println("<html><head></head><body>");
    out.println("<h1>" + strToken + "</h1>");
    out.println("</body>");
    out.println("</html>");
}
} // end doGet

```

The `getToken()` method creates a new `SametimeToken` object, passing it the user name as a parameter. It then calls the `SametimeToken`'s `getToken()` method.

Example 12-9 TokenGenerator's getToken() method

```

public String getToken(String strUserName) throws ServletException
{
    SametimeToken myToken = new SametimeToken(strUserName);
    return myToken.getToken();
}

```

SametimeToken class

The `SametimeToken` class is a private class within the `TokenGenerator` class, and contains all the Sametime-specific code. It implements three Sametime listener interfaces: the protected class `SametimeToken` implements `LoginListener`, `ResolveListener`, and `TokenServiceListener`.

The `TokenServiceListener` contains the methods that handle the token generation process. The class also defines a number of variables for the Sametime objects it uses (Example 12-10 on page 345).

Example 12-10 SametimeToken variables

```
// session object
protected STSession m_session;
// server application object
protected ServerAppService m_saService;
// lookup service object
protected LookupService m_lookupService;
// resolver object
protected Resolver m_resolver;
// sa token service object
protected SATokenService m_tokenService;
```

In the class' constructor, we create a new Sametime session and load the components we need into it. We then start the session and attempt to log in to the server.

Note: In the Sametime 3.1 Community Server Toolkit, the SATokenService component does not load if you use the loadAllComponents() or loadComponents() methods. You must load it explicitly as shown in Example 12-11.

Example 12-11 SametimeToken constructor

```
protected SametimeToken(String strUserName) throws ServletException
{
    m_strUserName = strUserName;

    try
    {
        m_session = new STSession("TokenGenerator " + this);
        String[] strNames = { ServerAppService.COMP_NAME,
LookupService.COMP_NAME };
        // load the ServerAppService and LookupService components
        m_session.loadComponents(strNames);
        m_saService = (ServerAppService)
m_session.getCompApi(ServerAppService.COMP_NAME);
        m_lookupService = (LookupService)
m_session.getCompApi(LookupService.COMP_NAME);
        // load the SATokenService component
        new SATokenComp(m_session);
        m_tokenService =
(SATokenService)m_session.getCompApi(SATokenService.COMP_NAME);

        m_session.start();

        login();
    }
}
```

```

        catch (DuplicateObjectException e)
        {
            System.err.println("Duplicate object exception");
            e.printStackTrace();
            throw new ServletException("Duplicate object exception");
        }
    }
}

```

The `login()` method in turn calls the `doLogin()` method. The `doLogin()` method attempts to log the servlet in as a server application. The default connection method is configured to connect through the Sametime mux, so we change this to connect directly to the server. We do this by explicitly setting the port number we wish to connect to (1516), and then calling the `setConnectivity()` method of the `ServerAppService` object. We specify our login type as `LT_SERVER_APP` for a server application, then add ourselves as a `LoginListener` before finally calling the `loginAsServerApp()` method. We then wait until we are logged in or until our login attempt times out. See for more details.

Example 12-12 SametimeToken doLogin() and isLoggedIn() methods

```

protected void doLogin(String m_strSametimeServer)
{
    // Login as a server application
    Connection[] connections = { new SocketConnection(1516, 17000),};
    m_saService.setConnectivity(connections);
    short loginType = STUserInstance.LT_SERVER_APP;

    m_saService.addLoginListener(this);
    m_saService.loginAsServerApp(m_strSametimeServer, loginType,
    "TokenGenerator", null);

    int nCount=0;
    loginExpired = false;
    // Loop until we have logged in or timed out...
    do
    {
        // Wait about 10 seconds (ie 100 x 100ms) for login to succeed
        // otherwise throw servlet exception
        if (++nCount > 100)
        {
            // Login attempt has timed out
            loginExpired = true;
        }
    }
    try
    {
        Thread.sleep(100);
    }
    catch (InterruptedException e)

```

```

        {
        }
    }
    while (isLoggedIn() == false && loginExpired == false);
} // end doLogin()

protected boolean isLoggedIn()
{
    if (m_saService == null)
    {
        return false;
    }
    return m_saService.isLoggedIn();
}

```

When the server application has logged in, the `loggedIn()` method is called. In Example 12-13, we create a `Resolver` object from the `LookupService` component, and remove ourselves as a `LoginListener`, as this is no longer required.

Example 12-13 SametimeToken loggedIn() method

```

// Event fired by successful login to server -
public void loggedIn(LoginEvent event)
{
    System.out.println("LoggedIn()");
    m_resolver = m_lookupService.createResolver(true, // Return all matches.
                                                false, // Non-exhaustive
lookup.
                                                true, // Return resolved
users.
                                                false); // Do not return
resolved groups.
    m_saService.removeLoginListener(this);
}

```

Control then passes back to the `getToken()` method (Example 12-14 on page 348) of the `TokenGenerator` object, and the `getToken()` method of `SametimeToken` is called, as shown in Example 12-9 on page 344. The `getToken()` method adds the class as both a `ResolveListener` and `TokenServiceListener`. It attempts to resolve the user name to a valid Sametime user by calling the `Resolver`'s `resolve()` method. It then waits for the boolean `m_bFinished` flag to be set to true before continuing. It begins cleaning up by removing itself as a `TokenServiceListener` and `ResolveListener` before logging out from the server. Finally, it stops and unloads the Sametime session before returning the token to the `TokenGenerator` object.

```
public String getToken()
{
    m_resolver.addResolveListener(this);
    m_tokenService.addTokenServiceListener(this);
    m_resolver.resolve(m_strUserName);
    try
    {
        int nCount=0;
        while (m_bFinished == false)
        {
            if (++nCount > 50)
            {
                m_bFinished = true;
            }
            else
            {
                Thread.sleep(100);
            }
        }
    } // end while
} // end try

catch (InterruptedException e)
{
}

m_tokenService.removeTokenServiceListener(this);
m_resolver.removeResolveListener(this);
m_tokenService = null;
m_resolver = null;

// Log out from the server...
m_saService.logout();

// Pause briefly to allow service to log out
try
{
    Thread.sleep(10);
}
catch (InterruptedException e)
{
}

// Stop the session and unload it
m_session.stop();
m_session.unloadSession();

return m_strUserToken;
```

```
}// end getToken
```

When the user is successfully resolved, SametimeToken's resolved() method (see Example 12-15) is called. We cast the resolved object into an instance of an STUser object, then pass it into the generateToken() method of the SATokenService object.

Example 12-15 SametimeToken resolved() method

```
// User is a valid Sametime user - generate a token for them
public void resolved(ResolveEvent event)
{
    System.out.println("resolved");
    if (event.getName().toLowerCase().compareTo(m_strUserName.toLowerCase())
== 0)
    {
        STUser portalUser = (STUser)event.getResolved();

        // Use the token service to generate a token for this particular STUser
object
        m_tokenService.generateToken(portalUser);
    }
}
```

For a successful token generation, SametimeToken's tokenGenerated() method (Example 12-16) is called. We get a handle on the Token object by calling the getToken() method of the TokenEvent object. We then check to make sure that the generated token is the one we have asked for by calling the getLoginName() method of the token object and comparing it with our user name. If the names match, we return the text value of the token by calling the getTokenString() method of the token object.

Example 12-16 SametimeToken tokenGenerated() method

```
// return the token as a string
public void tokenGenerated(TokenEvent event)
{
    System.out.println("tokenGenerated");
    Token theToken = event.getToken();
    if
(theToken.getLoginName().toLowerCase().compareTo(m_strUserName.toLowerCase())
== 0)
    {
        m_strUserToken = theToken.getTokenString();
        m_bFinished = true;
    }
}
```

}

Once deployed to a suitable servlet engine, Figure 12-4 shows the results of calling the servlet from a browser.

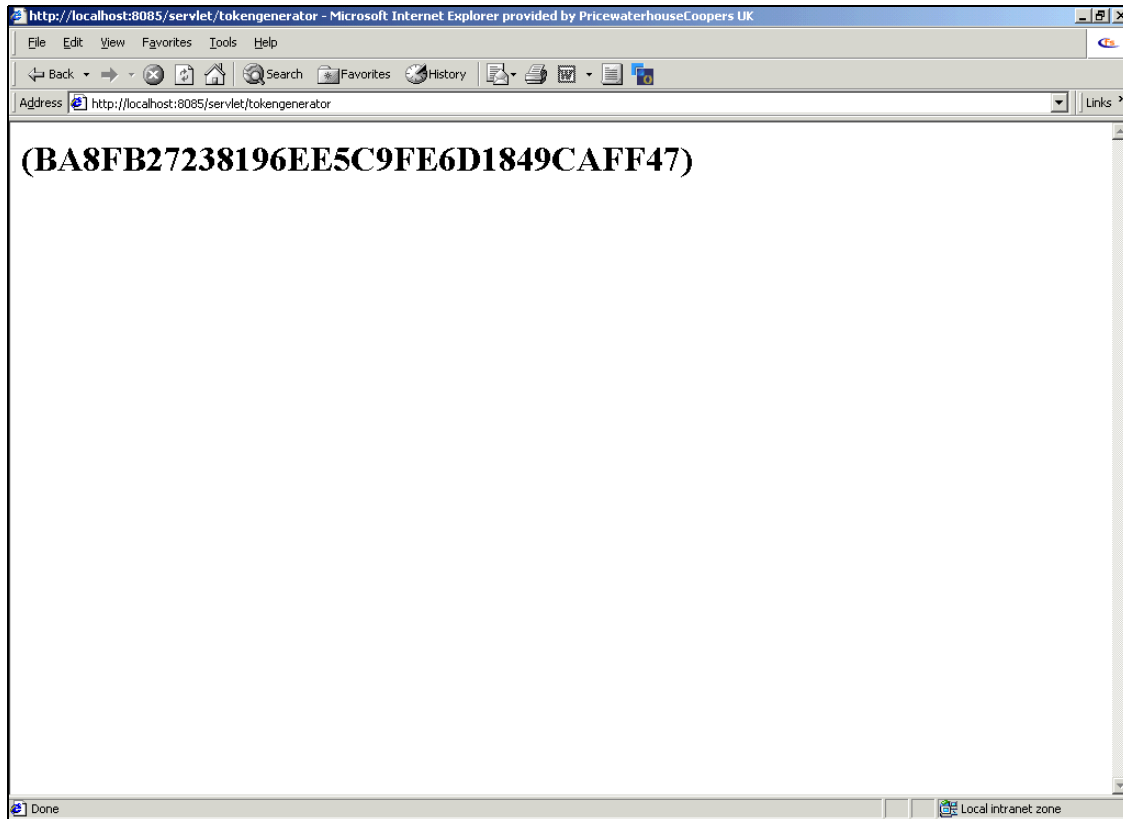


Figure 12-4 *TokenGenerator servlet's output*

Important: To allow the servlet to log in to the Sametime server as a server application, the IP address of the machine the servlet is hosted on must be in the Sametime server's trusted IP list. See 8.4.4, "Sametime server preparation" on page 191 for details on how to do this.

To prove that the token generated is valid and can be used to log in to the Sametime server, we can use a Java applet and the `loginByToken()` method. In the applet's `init()` method, replace the call to the `loginByPassword` method with a call to the `loginByToken()` method (Example 12-17 on page 351).


```
try
{
    m_session = new STSession("SampleApplet " + this);
}
catch (DuplicateObjectException doe)
{
    doe.printStackTrace();
    return;
}

m_session.loadAllComponents();
m_session.start();

m_communityService =
(CommunityService)m_session.getCompApi(CommunityService.COMP_NAME);
m_lookupService = (LookupService)
m_session.getCompApi(LookupService.COMP_NAME);
m_communityService.addLoginListener(this);
//m_communityService.loginByPassword("itsotest-st31.cam.itso.ibm.com",
"John Barrow", "password");
m_communityService.loginByToken("itsotest-st31.cam.itso.ibm.com", "CN=John
Barrow,O=Itsportal", "(BA8FB27238196EE5C9FE6D1849CAFF47)");
```

Note the format of the user name when the `loginByToken()` method is used, as opposed to the `loginByPassword()` method. The User ID, not any login name, must be used with the token service, as Sametime will not perform a resolve as it does for other logins using a password.

You must also use the token quickly, since it may expire after a set time period if the Sametime administrator has enabled an agent in the Secrets and Tokens database called `SametimeSecretGenerator`. This agent creates a new set of tokens at the specified interval. See the *Sametime 3.1 Administrator's Guide* for more information.

12.4 Enabling Active Server Pages (ASP) with Sametime Links

As discussed earlier in the redbook, Sametime Links is a toolkit that allows Web developers to Sametime-enable their applications. Most people realize that you can use the toolkit to Sametime-enable Domino, Notes, and WebSphere applications, but you can also use it on other Web application servers, such as Microsoft Internet Information Server (IIS). This section discusses an example of how to Sametime-enable a Microsoft Active Server Page (ASP) running on IIS.

12.4.1 What are IIS and ASP?

IIS is Microsoft's Web application server. It provides HTTP services as well as an environment for running ASP, which is its server-side scripting environment. You can use ASP to create dynamic Web applications much like applications developed using Domino or WebSphere Application Server.

Using the Sametime Links Toolkit, you can bring the power and collaboration features of Sametime to your Microsoft-based Web site.

12.4.2 Directories

The major issue faced when enabling an IIS/ASP with Sametime Links is how the directories for each server will be configured and integrated in your environment. Directories are a meaty topic that are beyond the scope of this redbook, but for this section, we assume that you have at least come to a point where both IIS and Domino can recognize a user by the same unique user name and that the Sametime directory contains a password for each user. Once you have this, you will be able to supply Sametime Links with the user name and password or token it needs to login.

Note: Our environment consisted on an IIS server listening on port 81 and Sametime/Domino sever listening on port 80. IIS used its own NT directory and Domino used an LDAP directory. The directories were not synchronized, but did use the same user names. The only reason we had this type of environment was because of limited time and resources.

12.4.3 Logging in using the Token Generator

Most Web sites now offer or at least try to offer a single sign-on experience. In our example, Sametime and IIS are using different authentication mechanisms, so we need a way to log users into Sametime without prompting them for their password again after they have already authenticated with IIS. To do this, we tweaked the Token Generator example in 12.3.1, "The TokenGenerator servlet" on page 342 to return the token in a JavaScript variable instead of an HTML page.

12.4.4 How it works

Figure 12-5 on page 353 shows the series of events when we try to load the Sametime-enabled Web page.

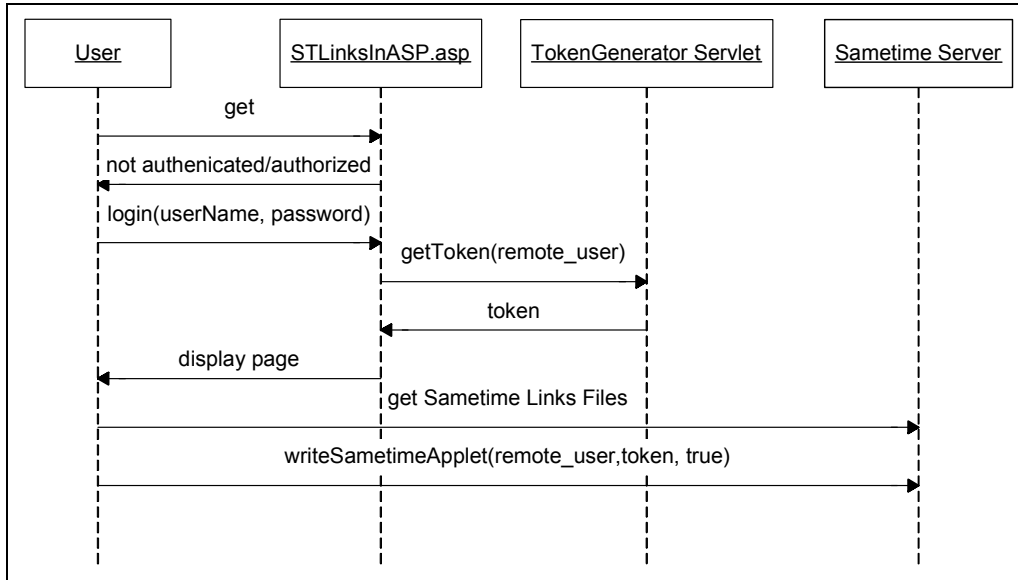


Figure 12-5 Sametime Links and ASP

The user requests to view the STLinksInASP.asp page, which has been Sametime-enabled. Since the page is access-protected by IIS, IIS will challenge the user to authenticate. The user then logs in with their user name and password for IIS. IIS authenticates them and then starts to dynamically generate the ASP page.

When generating the page, we first insert references to the Sametime Links stylesheet and JavaScript files, and then we call the setSTLinksURL JavaScript function. You will notice that we hardcoded the host name and directory in the page. See Chapter 5 of the *Sametime Links 3.1 Toolkit Developer's Guide and Reference* (see 2.2.1, "Sametime Software Development Kit (SDK) documentation" on page 22 for details on how to find this publication) for how to make this more dynamic.

Next, we open an HTTP connection to the TokenGenerator servlet to obtain the Sametime token for the authenticated user. (We assume in our example that this servlet has been protected so that only the IIS server can access it.) The code to accomplish this is shown in Example 12-18 on page 354.

Example 12-18 VBScript to call TokenGenerator servlet

```
<%  
Dim url  
Dim objHTTP  
  
Set objHTTP = Server.CreateObject("Msxml2.ServerXMLHTTP")  
objHTTP.setTimeouts 1000,1000,1000,1000  
  
url = "http://itsotest-st31.cam.itso.ibm.com/servlet/TokenGenerator?userName=" +  
+ Server.URLEncode(Request.ServerVariables("REMOTE_USER"))  
  
objHTTP.open "GET", url, False  
objHTTP.send  
  
if Err.number = 0 and objHTTP.status = 200 then  
    JS_ASP_SAMETIME_TOKEN = objHTTP.responseText  
else  
    JS_ASP_SAMETIME_TOKEN = ""  
end if  
  
Set objHTTP = Nothing  
%>
```

The rest of the page is basically static, except for when we dynamically insert the current user name into the writeSTLinksApplet() JavaScript function. Example 12-19 shows the rest of the HTML body.

Example 12-19 ASP HTML body

```
<h1>Using the Sametime Links Toolkit in an ASP Page</h1>  
<SCRIPT language="JavaScript">  
<%=JS_ASP_SAMETIME_TOKEN%>  
writeSTLinksApplet('CN=<%=Request.ServerVariables("REMOTE_USER")%>,0=Itsportal',  
    ASP_SAMETIME_TOKEN , true);  
writeSametimeLink('Jonas Covey','Jonas Covey', true);document.write('<br>');  
writeSametimeLink('John Barrow','John Barrow', true);document.write('<br>');  
writeSametimeLink('Washington Cabral','Washington Cabral',  
    true);document.write('<br>');  
  
function STLinksLoggedIn(myUserID, myDisplayName) {  
    alert('logged in as:\n\n ' + myUserID + '\n\n ' + myDisplayName);  
}  
function STLinksLoggedOut(reason) {  
    alert('logged out: ' + reason);  
}  
</SCRIPT>
```

Tip: In order to use the common name in the writeSametimeLink JavaScript function, we had to change a Sametime server setting to allow users to resolve user and group names. Without changing this setting, we had to specify the fully distinguished name of the user (that is, CN=Jonas Covey,O=Itsportal). We found that CN and O had to be capitalized or else it did not work.

When all this is complete, the page is sent to the user. Their browser then parses and loads the HTML. Figure 12-6 shows the Sametime-enabled ASP.

Using the Sametime Links Toolkit in an ASP Page

■ [Jonas Covey](#)

- John Barrow
- Washington Cabral

Figure 12-6 Sample Sametime-enabled ASP

Note: This example is really a proof-of-concept to show that you can Sametime-enable an ASP page. Obviously, there are many improvements that could be made to make it more efficient. For example, the first time the ASP generates the Sametime token you could place the value in a cookie. This way, you would not have to generate the token with each page refresh.

We employed a software/programming level approach to our solution. Infrastructure level solutions are also possible; however, they were beyond the scope of this chapter.



Part 3

Bringing it together

In this part, we provide a business context to illustrate the possible uses of Sametime.



Visioning scenario: Sametime enterprise integration

This chapter provides a business and architectural context that illustrates how many of the code examples in earlier chapters can be incorporated into a “real world” scenario. Most of the technical approaches to customizing and enhancing Sametime functionality described previously in the redbook are brought together in a business context. Ultimately, we hope this will help you better appreciate how your organization may benefit by more effectively leveraging Sametime.

We start by giving an overview of a typical company who wants to build new collaborative capabilities for both its internal employees and its external customer relations process. We describe the business drivers and discuss where and how Sametime can help improve efficiency.

While this overall scenario is a fictitious one, the examples are closely modeled after real examples and experiences from the field. The goal is to provide a realistic but intentionally general scenario that may be applied to a broad range of businesses. Finally, please note that the intent of this chapter is not to focus on specific deployment issues.

Attention: This purpose of this chapter is expand your thoughts about instant collaboration within a business context. It is not intended to focus on detailed specifics of implementation. Accordingly, some of the technical details and approaches provided are discussed at a high level. Within each section, you will find references to specific chapters that highlight more detailed implementation techniques

13.1 The scenario

The scenario that we are about to describe is based on a medical equipment manufacturing company called Fictitious Medical Equipment Inc. (FME)

While this is not based on an actual company, the business examples provided have been designed to fit a broad range of company profiles, from small and medium businesses to larger enterprise organizations. Where appropriate, we will be highlighting any industry specific possibilities.

In addition to manufacturing medical equipment, FME provides the following services:

- ▶ Online sales for parts and accessories.
- ▶ Custom designed medical equipment to fit individual patient and hospital needs.
- ▶ Consulting services to design hospital solutions for imaging equipment.
- ▶ Technical support for equipment operational issues.
- ▶ Training for customers on operating their equipment.
- ▶ Developing new equipment to address market need and remain a leader within the industry.

The technical aspects of the scenario focus primarily on FME's call center interface. This interface is linked closely to each of the departments within the scenario. It is the primary exchange for providing information and support to customers, visitors, and internal employees.

13.2 Business drivers and requirements

FME hopes to address the following business needs in order to better satisfy their customers. Integrating Sametime throughout the organization can help satisfy each of these requirements:

- ▶ Resources from within different departments need more effective real-time collaboration tools to resolve customer issues more effectively.
- ▶ Customers must be able to address most of their own needs through self service tools on Fictitious's Web site.
- ▶ The costs of their call center infrastructure are too high, since most of the customers requests come through phone calls. FME needs to provide an alternative channel to allow customers to interact with their sales team, consultants, and engineers.

- ▶ A broad set of applications has been deployed throughout the different departments. Each of these applications has a unique interface with the call center. Working with these many applications requires extensive training, complex IT integration, and ultimately reduces employee productivity. FME wishes to move toward a common suite of applications.
- ▶ Sales people are tied to common customer tracking tools.

FME's external customers have also expressed the following requirements:

- ▶ Faster response times for issues they post on the Web site.
- ▶ Helpful tools to address their need on clarifying some products technical data.
- ▶ Provide more personalized relationships with their representatives.
- ▶ Easier access to technical staff and sales representatives.

13.2.1 Relationships between the call center and the departments

Figure 13-1 on page 363 illustrates the relationship between the call center, the customers, and other internal departments. It also describes each of the primary customer and departmental activities.

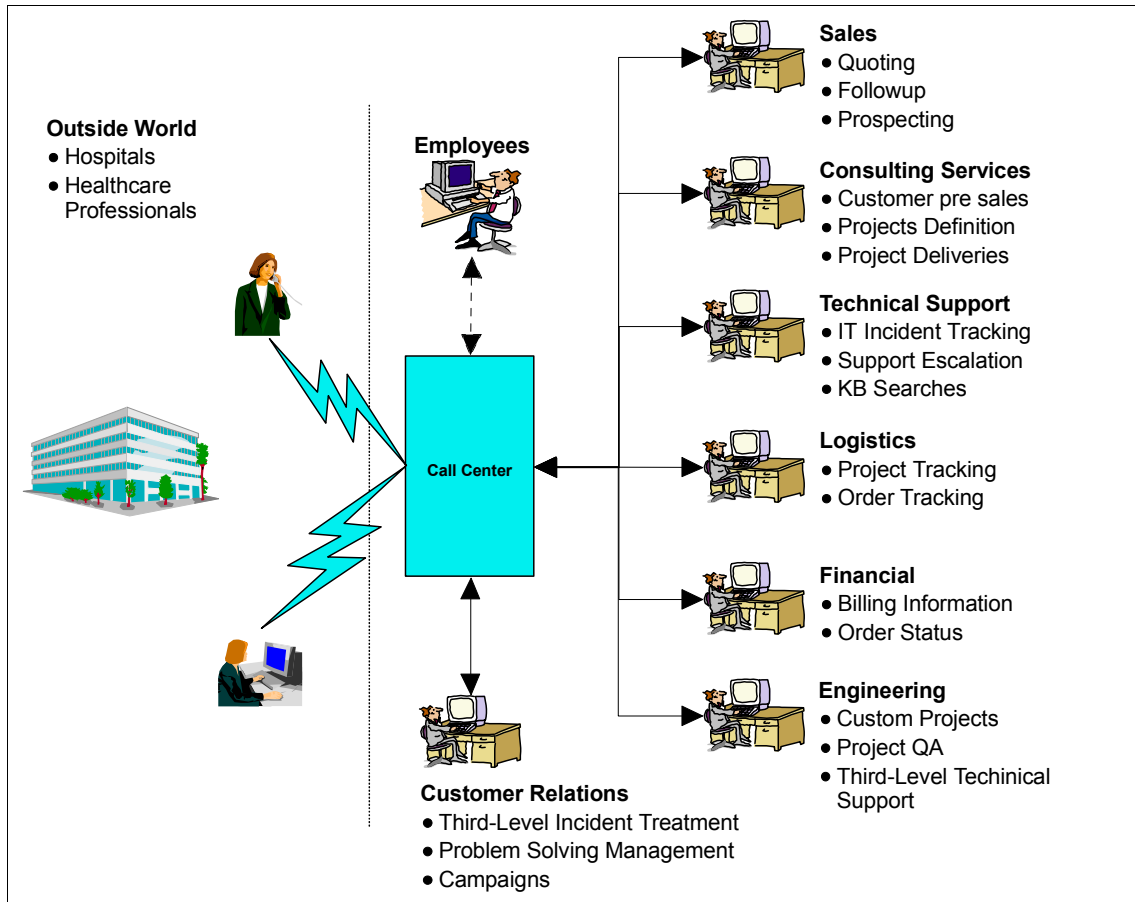


Figure 13-1 Diagram illustrating relationships in the business scenario

Note that Figure 13-1 illustrates the existing relationships before the integration of Sametime into the organization. As Sametime capabilities are integrated, the diagram will be modified to illustrate more direct channels of communication.

- The outside customer interface is based on the company's Web site functionality. Current capabilities allow customers to review product data, search for tech notes in knowledge bases, and send e-mail to FME employees.
- Each department has its own applications with unique, non integrated user interfaces.

According to the diagram in Figure 13-1, we now describe the roles and responsibilities of each of the actors and departments within the scenario.

External customers

FME’s customers are primarily hospitals, health care professionals, and medical equipment distributors.

Among other things, the most common requests from Fictitious customers are:

- ▶ Quoting for pre-designed equipment and accessories
- ▶ Online purchase of accessories
- ▶ Questions solving with sales representatives
- ▶ Technical support on using equipment
- ▶ Requests for sizing services for medical environments

Employees (internal customers)

FME has designed their call center structure to have the company’s employees address their needs through the same call center infrastructure as the one created for outside customers. They can use this channel to post questions related to IT, HR, and so on.

Departmental roles and responsibilities

Table 13-1 describes the different departments within FME and their primary responsibilities. While it is important to understand the role of each department, consider also the channels of communication between each department and the call center. Integrating Sametime into the organization will allow for a much greater degree of direct communication between departments.

Table 13-1 FME’s departments

Department	Roles and responsibilities
Customer Relations Department (CR Department)	<ul style="list-style-type: none">▶ Dealing with customer satisfaction issues and overall customer relationship management.▶ Employees are treated as inside customers, so any employee satisfaction issues ultimately become the CR department’s responsibility as well.▶ Tracking down customers’ needs based on their buying behavior and the types of incidents reported.

Department	Roles and responsibilities
Sales Department	<ul style="list-style-type: none"> ▶ Providing customers with quotes for specific products or solutions. ▶ Determining forecasts and maintaining sales pipelines. ▶ Selecting products or services that can be offered to existing or potential customers. ▶ Communicating with the customer relations department about potential campaigns to increase sales or to properly educate customers.
Consulting Services	<ul style="list-style-type: none"> ▶ Designing and offering customers, solutions designed with Fictitious equipment to drive specific needs. ▶ Training customers on how to properly operate equipment.
Technical Support	<ul style="list-style-type: none"> ▶ Resolving technical issues for internal employees and external customers ▶ Resolve incidents related to B2C and B2B features of the company's Web site ▶ Educate customers on the issues and questions related to medical and computer equipment operation. ▶ Build and refine IT knowledge bases and to orient employees and customers about where to find the desired information. ▶ Deal with technical support issue escalations when necessary. This process can consist of a simple incident, or a request for a new feature within the companies online capabilities.

Department	Roles and responsibilities
Shipping and Logistics	<ul style="list-style-type: none"> ▶ Handle the flow of services and products. ▶ Provide customers with the proper information about products and services delivery. ▶ Tracking and reporting on customer order status. ▶ Coordinate efforts between different company departments to provide consistent information to the customer about a specific request.
Financial	<ul style="list-style-type: none"> ▶ Provide billing status about orders, customer accounts, and so on. ▶ Inform the customer relations department about the most common problems related to financial order processing.
Engineering	<ul style="list-style-type: none"> ▶ Developing new equipment. ▶ Enhancing the existing installed equipment base. ▶ Designing custom equipment. ▶ Interact with the technical support department in order to solve a sever customer problem.

Now that the departments and their relationships to each other and the call center have been more clearly defined, we will define the business needs to be met by integrating Sametime throughout FME.

13.2.2 Solution description

The approach to the solution is based on two principles:

- ▶ Maintain and continue investments in existing application development efforts, enhancing these applications to take advantage of Sametime instant collaboration features.
- ▶ Consolidate the user interface into a consistent Web-based UI, while still keeping the original applications. Ultimately, a consistent UI, enabled with real-time collaborative capabilities, will result in a more productive workplace.

Figure 13-2 illustrates an overview of the business logic architecture. The grey shadowed box surrounding the User Interface Layer and the Business Applications Layer represents a context for the proposed scope of integrating Sametime real-time collaborative capabilities. Finally, the bidirectional arrow at the bottom of the diagram indicates the channel of information flowing between all layers of the architecture.

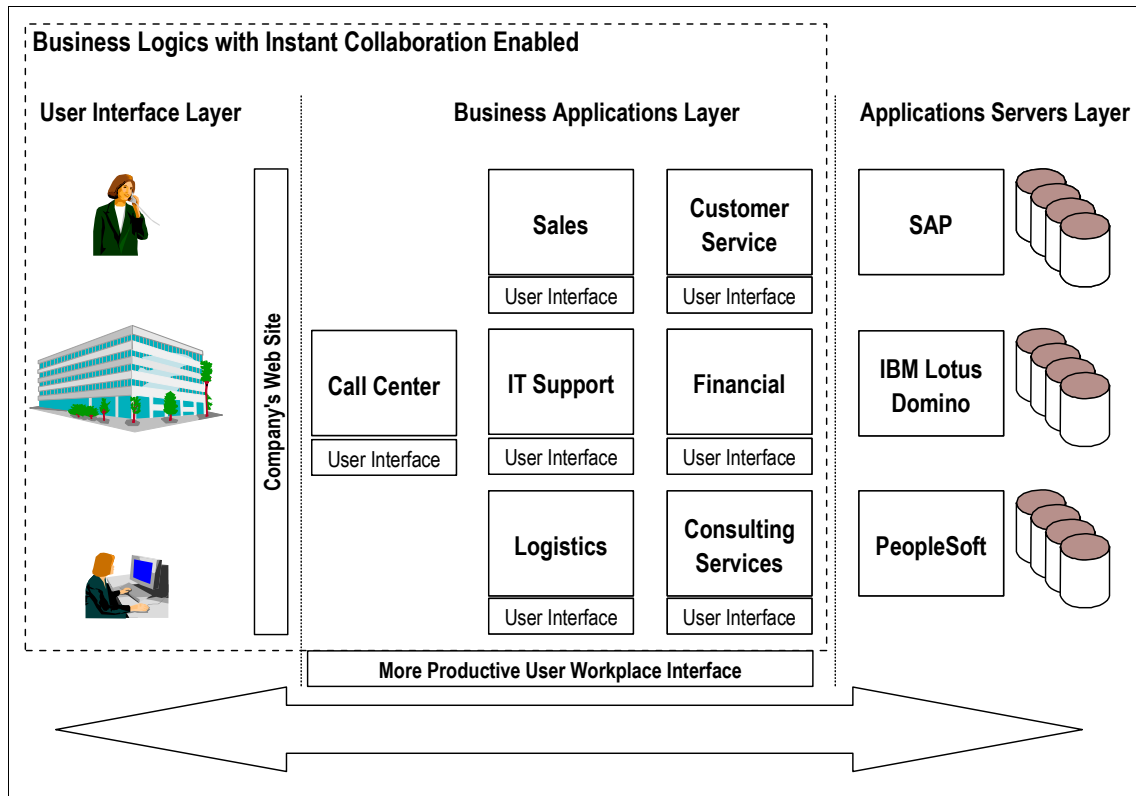


Figure 13-2 Illustrates collaboration approach to current environment

13.3 Architectural considerations

Since the objective of this redbook is not to cover specific aspects of Sametime deployment, the information provided in this section is intentionally discussed at a very high level. We do not discuss any hardware issues in detail, but mention only the software aspects of the solution.

13.3.1 Software components

In this section, we describe the software components used to build FME's solution. Each software component contains a brief description of its functionality within the whole solution (see Table 13-2).

Table 13-2 Software components for FME's solution

Software component	Function
IBM Lotus Domino Server Version 6.02	Used within Fictitious as the messaging infrastructure and for applications such as customer tracking, incident manager, and knowledge bases
IBM Lotus Instant Messaging and Lotus Web Conferencing (Sametime) Version 3.1	Implemented as the instant collaboration framework for the solution. The following sub components are used: <ul style="list-style-type: none">▶ Java Toolkit▶ C++ Toolkit▶ COM Toolkit▶ Sametime Links Toolkit▶ Community Server Toolkit▶ Directory and Database Access Toolkit▶ Chat Logging API
IBM WebSphere Portal Server Version 4.2	Serves as the framework for interface consolidation.

Operating system considerations

Since Sametime deployment is not the primary focus of this redbook, we do not cover specific information about the operating system used for FME's solution.

However, we do wish to illustrate the supported operating systems for each of the software components discussed in the solution. Figure 13-3 on page 369 illustrates the supported operating system matrix.

Table 13-3 Supported platforms matrix

Software	1	2	3	4	5	6	7	8	9
IBM Lotus Domino Version 6.0.2	X	X	X	X	X	X	X	X	X
IBM Lotus Instant Messaging and Lotus Web Conferencing (Sametime) Version 3.1	X	X	X ^a	-	X ^a	-	X	-	-
IBM WebSphere Portal Server Version 4.2	X	X	X	-	X	X	-	-	X

a) Coming soon.

Where:

1. Microsoft Windows NT® 4.0 operating system (using an Intel® processor)
2. Microsoft Windows 2000 operating system (Server and Advanced Server editions)
3. Sun Solaris Operating Environment 8/SPARC
4. Sun Solaris Operating Environment 9/SPARC
5. IBM AIX® operating system, Version 4.3.3x
6. IBM AIX® operating system, Version 5.1
7. IBM OS/400® V5R1 or later, using the IBM @server iSeries (formerly IBM AS/400®)
8. IBM OS/390® V2R10 or later, using the IBM @server zSeries® (formerly S/390®)
9. Red Hat Linux®, Version 7.2 or SuSE Linux, Version 8.0 (using an Intel processor)

Attention: At the time of writing this redbook, Table 13-3 represents the supported platform matrix. Please refer to the software support sites for any updated information.

13.4 Phased approach toward building the solution

In the upcoming sections, we describe phases and specific steps taken toward building the solution. The overall implementation of the solution has been broken down into three phases (see Table 13-4 on page 370).

Table 13-4 Objectives of each implementation phase

Phase	Objectives
Phase I	<ul style="list-style-type: none">▶ Building/Installing the underlying Sametime Infrastructure (Servers and clients).▶ Customizing the Meeting Center to more closely match FME's brand image
Phase II	<ul style="list-style-type: none">▶ Enabling People awareness in existing applications▶ Enabling offline message delivery▶ Implementing workflow based on the presence awareness
Phase III	<ul style="list-style-type: none">▶ Providing outside customers with "self service" tools▶ Enabling a bot to automatically respond to FAQ's▶ Enabling outside users to interact with internal customer representatives through online/chat dialog

13.5 Phase 1: Implementing the infrastructure

In this first phase, the goal is to implement the basic Sametime infrastructure throughout FME. Once the basic Sametime infrastructure is in place, including the ability for instant messaging and Web conferencing, Phase I also includes basic customizations to the meeting center.

13.5.1 Implementing an infrastructure for instant collaboration

One of the business needs identified in 13.2, "Business drivers and requirements" on page 361 was to implement tools to enable instant collaboration between people of different divisions.

To provide this functionality, FME will install the Sametime 3.1 server and the Sametime client. At this point, the installation is a straightforward, out of the box installation without any modifications. Sametime can be linked to either a Domino directory or an LDAP directory. The administrator will specify the configuration at the time of the installation.

With those standard features, we are able to offer the following functionalities:

- ▶ Instant messaging collaboration
- ▶ People awareness
- ▶ Audio and video conference
- ▶ Application sharing

In addition to providing FME with the functionality shown above, FME has now also established the base infrastructure for the rest of features to be implemented.

Implementation methodology

As mentioned earlier, deploying a Sametime 3.1 server and the Sametime 3 client with its out-of-the-box features will provide FME with functionality for instant messaging, Web conferencing, and other real-time collaborative capabilities. Since it is outside the scope of this redbook to provide specific instructions on installation and deployment, we will not provide details about how to install a Sametime Server. The resources shown in Table 13-5 provide in-depth information about installing, configuring, and deploying a Sametime server.

Table 13-5 Helpful resources for installing an configuring Sametime

Resource	Information
http://www.lotus.com/sametime	General marketing information about the product, which will help you start planning your implementation.
Lotus Developer Domain - Sametime Product Page (http://www.lotus.com/1dd)	You will be able to find documentation related to implementing Sametime as well as developing techniques. Follow the "Documentation" link to find installation and deployment related information: <ul style="list-style-type: none">▶ Sametime Installation Guide for different platforms▶ Sametime Administrator's Guide▶ Sametime User's Guide▶ Sametime Quick Start Guide

The results

Figure 13-3 on page 372 illustrates some of the out-of-the-box Sametime features that can be used. Each user who has the Sametime Connect Client installed on their desktop can use the buddylist to chat with other colleagues. Presence Awareness and User Status features are incorporated into the buddylist to illustrate which users are logged in and whether or not they are

available. When appropriate, users may initiate an n-way chat to communicate in real time between multiple users.

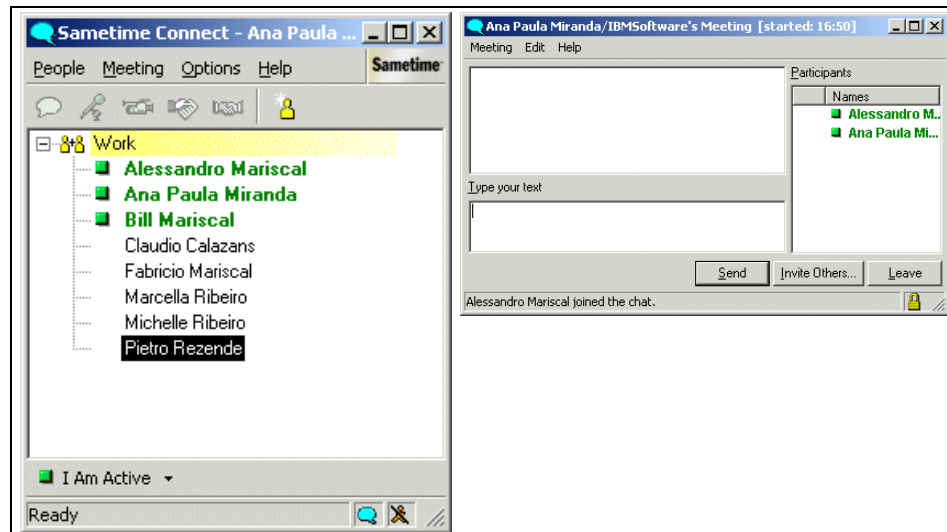


Figure 13-3 IBM Lotus Instant Messaging (Sametime) Connect Client

Sametime Online Meeting or Conferencing Services provide the ability to share objects (such as desktop applications, presentations, documents, and drawings) online. Users can schedule an online meeting in advance, or move directly from an instant message to a screen-sharing or whiteboard session. Figure 13-4 on page 373 illustrates the Sametime functionality for whiteboard and application sharing.

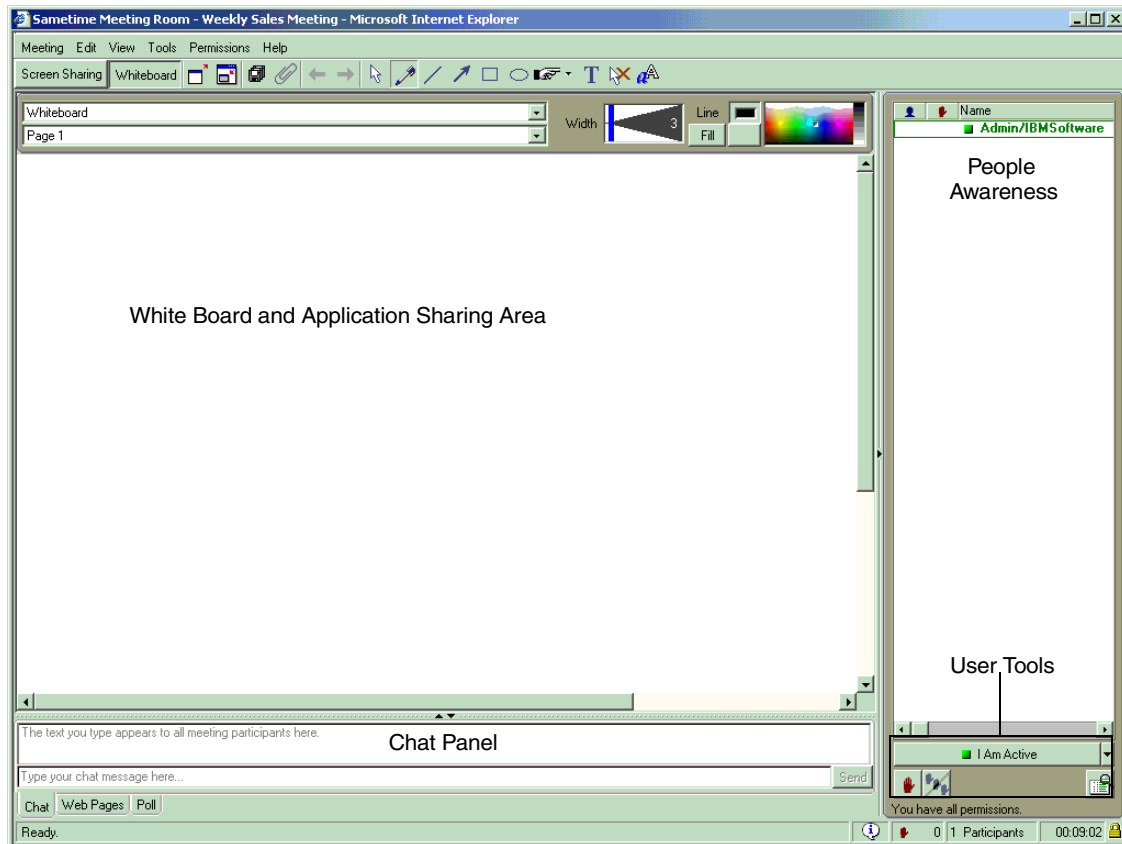


Figure 13-4 Application Sharing functionality

13.5.2 Customizing the meeting center look and feel

At this point, the assumption is that the Sametime 3.1 server has been installed and configured successfully. However, the standard look and feel of the Sametime Meeting Center does not match FME's corporate intranet design standard. Both the initial logon screen for entering the meeting center as well as the corporate logo and menus in the top and left frames need to be modified to properly brand the meeting center.

Figure 13-5 on page 374 illustrates an example page from FME's corporate intranet site. Notice that their logo and name span the top frame, while the left frame contains specific links to FME resources. The objective is to now modify the Sametime meeting center to more closely match this design standard.

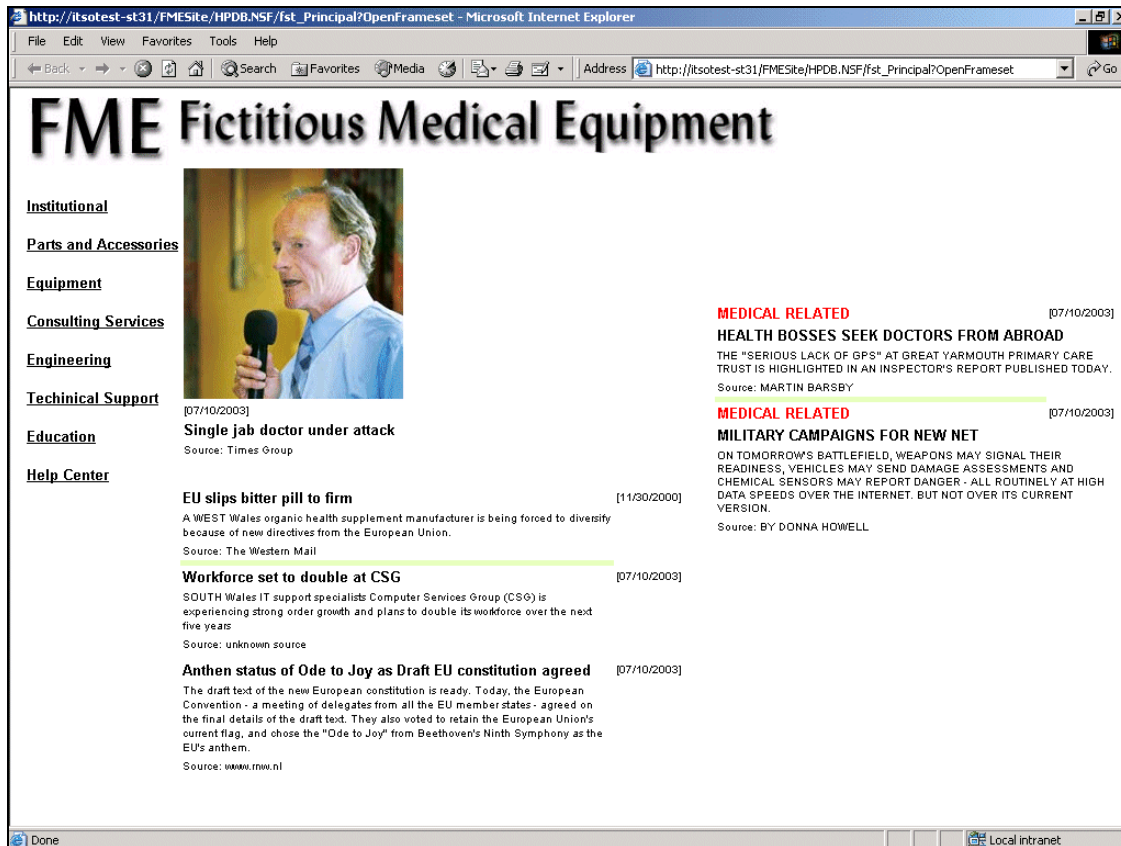


Figure 13-5 An example page from FME's intranet

Methodology for customizing the meeting center

We will be focusing on the following aspects of customizing the Sametime Meeting Center:

- ▶ Customizing the logon screen for the Sametime Meeting Center
- ▶ Sametime Meeting Center background images and colors
- ▶ Sametime Meeting Center menu options and colors
- ▶ Administration Panel menu options format and colors
- ▶ Including FME's logo on every page

In order to accomplish these tasks, we used the techniques discussed in the Chapter 11, "Customizing the Online Meeting Center" on page 297. Please refer to this chapter for additional technical guidance if necessary.

Customizing the logon screen

Upon installing the Sametime server into the environment, the logon screen is changed to one residing on a Sametime database called Sametime Center (stcenter.nsf).

By default, the initial Sametime logon screen has the Lotus Sametime corporate logo. Within the context of FME's intranet site, the Sametime standard logon screen initially looks like Figure 13-6.

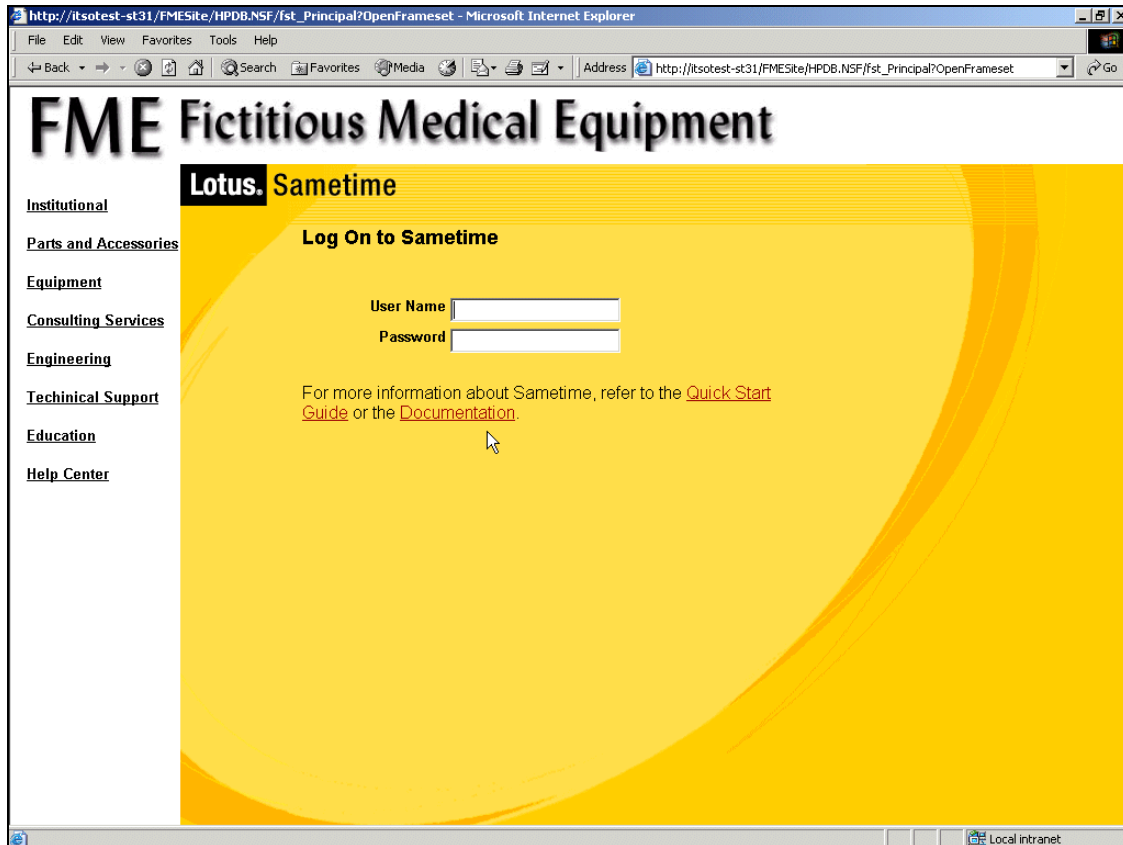


Figure 13-6 Sametime default logon screen within FME site

To make the logon screen adhere to FME's intranet design standards, we decided to perform the following modifications:

- ▶ Substitute the Lotus Sametime logo for the "Web Logon" logo used by FME.
- ▶ Change the background to a white background.

- Change the text above the user name prompt from “Log On to Sametime” to “Please, log on”.

Since the logon screen is a form inside a Notes database (stcenter.nsf), we need the Lotus Domino Designer client to accomplish this customization.

The specific form is named STLogonForm. Even though it is Domino based form, we are able to perform the customizations within this form mostly based on pure HTML skills.

The following steps were taken to customize the logon screen:

- Substituting the Lotus Sametime logo

Opening the stcenter.nsf database, we are able to find a view called All Pages. Within this view, there is a document titled Graphics. Inside this document, on the gifs section, there is a file called sametimelogo.gif.

Changing this file for another one with the same name, but with the Web Logon graphic, creates the desired effect.

Important: This is the fastest way to change the logo. However, we can attach a new file within this the Graphics document and then reference this new file on the STLogonForm under the HTML tag:

```
<IMG SRC="/pages/Graphics/$FILE/sametimelogo.gif" alt="" title=""  
border="0">.
```

Change the sametimelogo.gif for the name of the new attached file and the results will be the same.

- Changing the background

In this form, the background color and image attributes are used. The background image is found by a CSS reference to the file stbackground.gif, located in the directory sametime/images, under the domino/html directory.

To remove the background image, we changed the file name within the CSS tag to one that does not exist. The tag is located under the HTML Head Content attribute of the STLogonForm.

To change the background color, we changed the value of the CSS tag background-color: to #FFFFFF.

- Changing the static text was just a typing effort within the form. Finding the original text and replacing it with the one we wanted gave us the desired result.

Important: Customizing the default logon page, as well as any customization, is a development effort. Hence, the techniques used can vary according to one's skills or patterns.

Also, if you already have a session based customized logon page, you can keep it by simply pointing the domcfg.nsf configurations to the existing page.

Customizing the meeting center

Similar to the changes required for the Sametime default logon screen, the Meeting Center also does not adhere to FME's intranet site visual design standards. Accordingly, we had to customize its appearance to fit as well.

Since the technique for customizing the Meeting Center is covered in significant detail in Chapter 11, "Customizing the Online Meeting Center" on page 297, we do not present each of the detailed technical steps taken to perform this customization in this section. We present the highlights of the approach and recommend the reader to refer to Chapter 11, "Customizing the Online Meeting Center" on page 297 for the more detailed technical approach.

Essentially, the customizations for the meeting center focused on similar requirements as the logon page customization:

- ▶ Remove the background for a white background
- ▶ Adjust some font types
- ▶ Include some references to the company logo

Important: Please be aware that in the event of upgrading your Sametime server, you will need to review and document the design changes you have made to the Sametime Meeting Center.

By default, design changes will not be carried over to upgraded installation. Additionally, it is strongly recommended that each customization is thoroughly tested on an upgraded server in a test environment *prior* to deploying the changes into production.

The results

After completing the required branding design changes, FME's Sametime logon screen and Meeting Center have been rebranded to look like the example shown in Figure 13-7 on page 378.

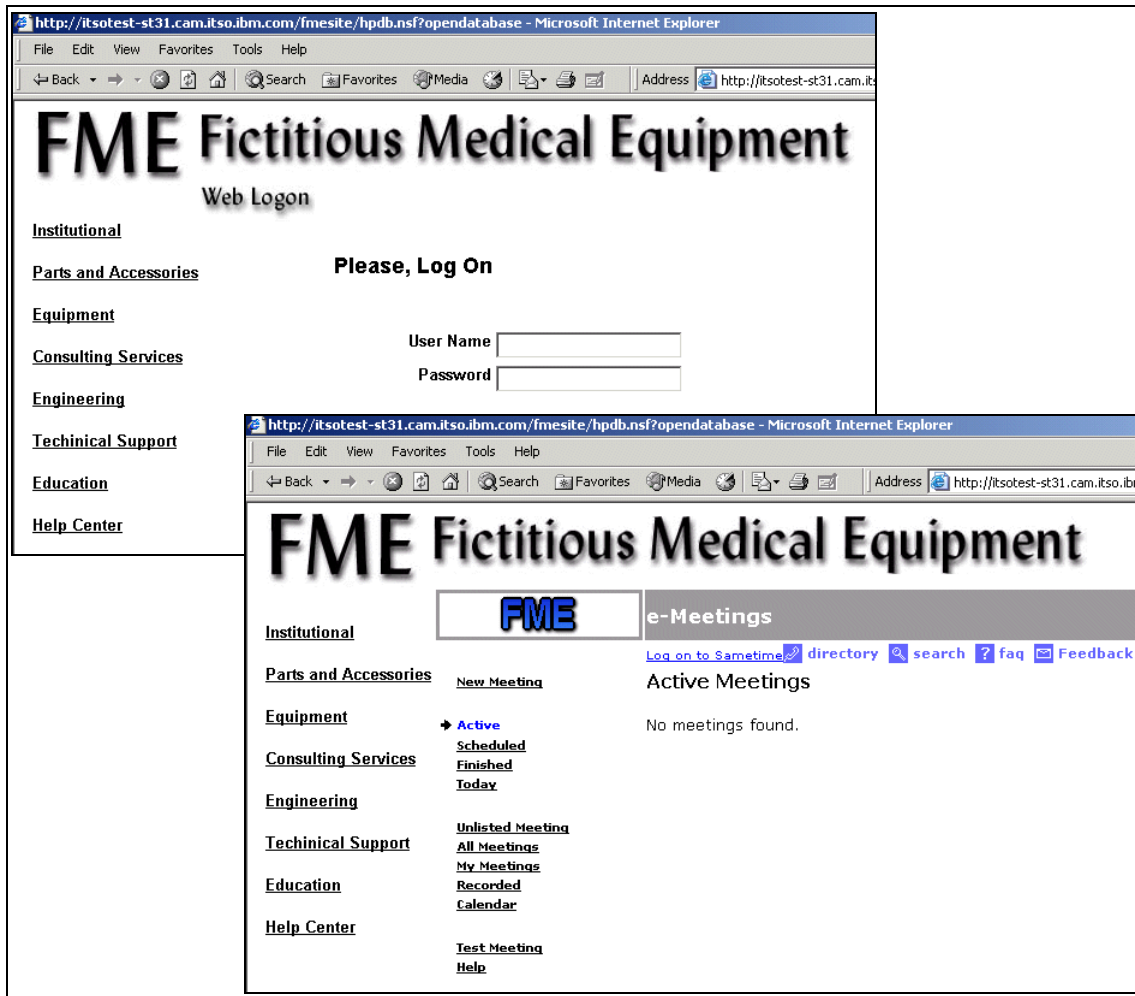


Figure 13-7 FME's customized Sametime logon page and Meeting Center

13.6 Phase 2: Expanding Sametime features

Having completed the first phase, the assumption going forward is that the Sametime infrastructure is functional. Sametime's core functionality, including instant messaging capabilities and Web conferencing, is now providing FME with a basic level of real-time collaborative capability.

In this next section, we introduce a more advanced level of Sametime integration into FME's existing business processes. Within each of the following sections,

we will discuss how to integrate presence awareness into existing applications and integrate Sametime functionality into FME workflow applications. The following section highlight integrating the following capabilities:

- ▶ Section 13.6.1, “Enabling people awareness into existing applications” on page 379
- ▶ Section 13.6.2, “Allow offline message delivery” on page 383
- ▶ Section 13.6.3, “Implement workflow using Sametime awareness feature” on page 390

13.6.1 Enabling people awareness into existing applications

One specific feature that has been strongly requested by most of the units within the company is to provide people awareness within existing applications. By enabling people awareness in applications, this allows FME employees to collaborate in the following ways:

- ▶ See if the authors of a specific document are online and available to collaborate
- ▶ Initiate instant messaging and Web conferences with other online colleagues, within the same context of a specific application
- ▶ Identify who is using the same document you are

Here are some of the specific business unit requests:

- ▶ Enable the customer tracking application with people awareness. This would make it easier for call center attendants to identify and collaborate with other colleagues related to a specific customer.
- ▶ The engineering department would like to have the incident tracking application enabled with people awareness. This feature would make it easier to identify additional technical support people involved on specific incidents and then collaborate to solve the issue more efficiently.
- ▶ The manager of the financial unit wishes to enable presence awareness to more effectively collaborate and communicate with the many different people involved in each customer invoice. Questions can be resolved quickly.
- ▶ Finally, presence awareness within the logistics department application will facilitate communication and further ensure consistent and reliable service.

Ultimately, each business unit’s goal is to increase productivity through instant messaging, presence awareness, and real-time collaborative capabilities.

Methodology for implementing presence awareness

To illustrate how to accomplish this, we will focus on the customer tracking application. This application keeps records of all of FME's customers. It also serves as a repository for follow-up items, such as phone calls, correspondence, and customer meetings notes. Each entry within the database has its author and other relevant participants' names on it. The application user interface is Web based. Figure 13-8 shows the actual FME Customer Tracking application. Notice there are user names available within most of the different fields.

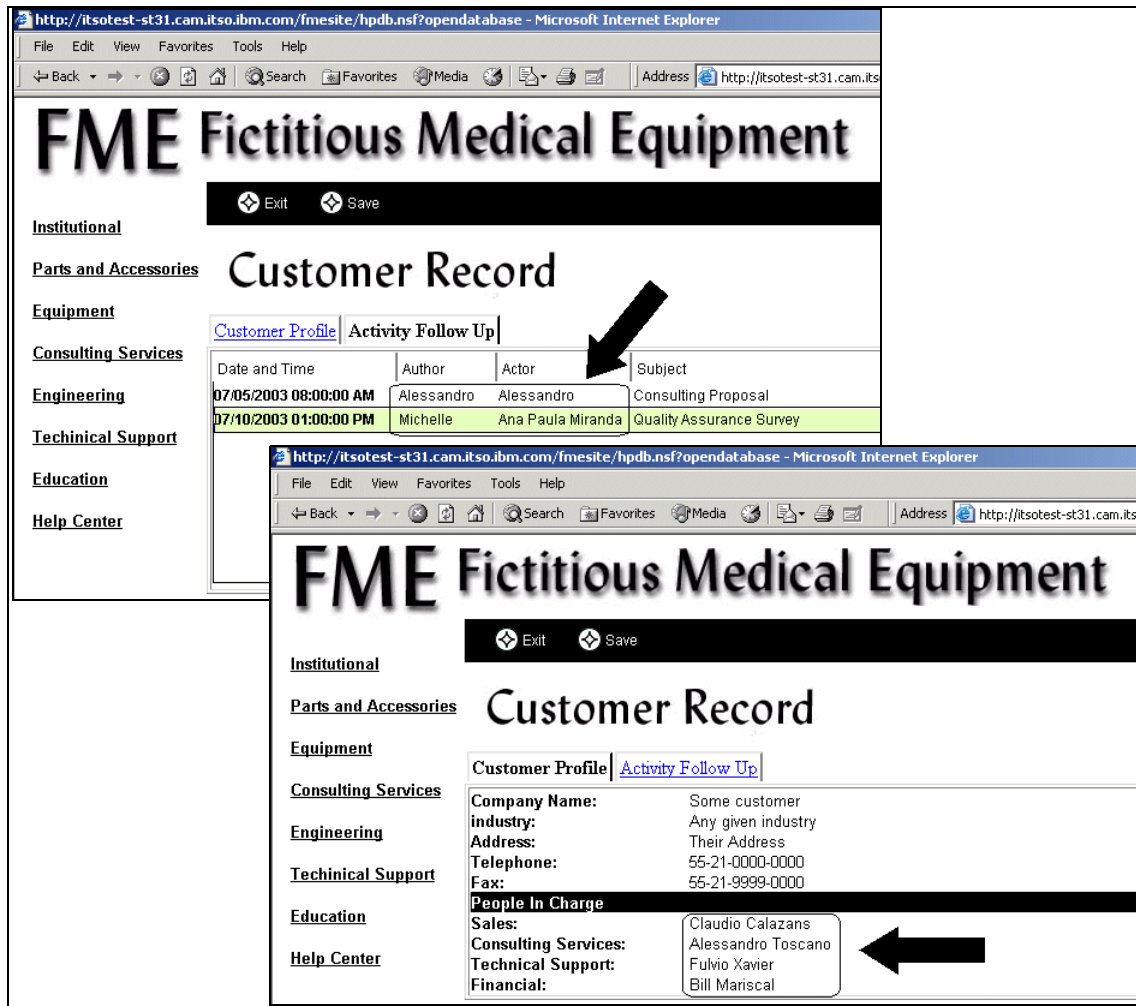


Figure 13-8 Actual FME customer tracking application

Presence awareness is implemented using the Sametime Links Toolkit. Since the Customer Tracking application uses a Web-based interface, the Sametime Links Toolkit is the best choice to accomplish the request.

Note: Although “Enabling the customer tracking application” on page 381 outlines a high level process for implementing people awareness using the Sametime Links Toolkit, please also refer to Chapter 9, “Sametime Links” on page 219 for more detailed information on how to implement awareness using this toolkit.

Enabling the customer tracking application

The following steps were performed to implement presence awareness using Sametime Links into the application:

1. In the HTML HEAD tag of the customer record form, we added the code to set the necessary parameters to call Sametime Links, as shown in Example 13-1.

Example 13-1 HTML HEAD tag to call Sametime Links

```
<HEAD>
...
<LINK REL=STYLESHEET HREF="http://<st server>/sametime/stlinks/stlinks.css"
TYPE="text/css">

<SCRIPT src="http://<st server>/sametime/stlinks/stlinks.js"></SCRIPT>

<SCRIPT>
    setSTLinksURL("http://<st server>/sametime/stlinks","en",
    "http://<st server>/sametime/stlinks")
</script>
...
</HEAD>
```

Note that the `<st server>` parameter in Example 13-1 represents the host name of the Sametime server used for enabling the application (for example, `itsotest-st31.cam.itso.ibm.com`).

2. Next, we had to call the Sametime Links applet and pass the necessary login information, as shown in Example 13-2.

Example 13-2 Call to the Sametime Links Applet with logon information

```
<script>
    writeSTLinksApplet("<user name>","<password>",<is by token flag>);
</script>
```

By integrating this with your Web application server, the user name and password properties can be passed as a token. Without this, the user would need to provide the logon information again.

3. Finally, for each field where user names are shown, we included the `writeSametimeLink()` function, as shown in Example 13-3.

Example 13-3 writeSametimeLink() function

```
<script>
writeSametimeLink("<user id>", "<display name>", <resolve flag>, "<options>");
</script>
```

In summary, include references to the Sametime link Toolkit, using HTML tags where names of people appear.

The results

After completing the development steps outlined in "Methodology for implementing presence awareness" on page 380, the user names within the Customer Tracking application appear as "live names", indicating online user status. The results are shown in Figure 13-9 on page 383.

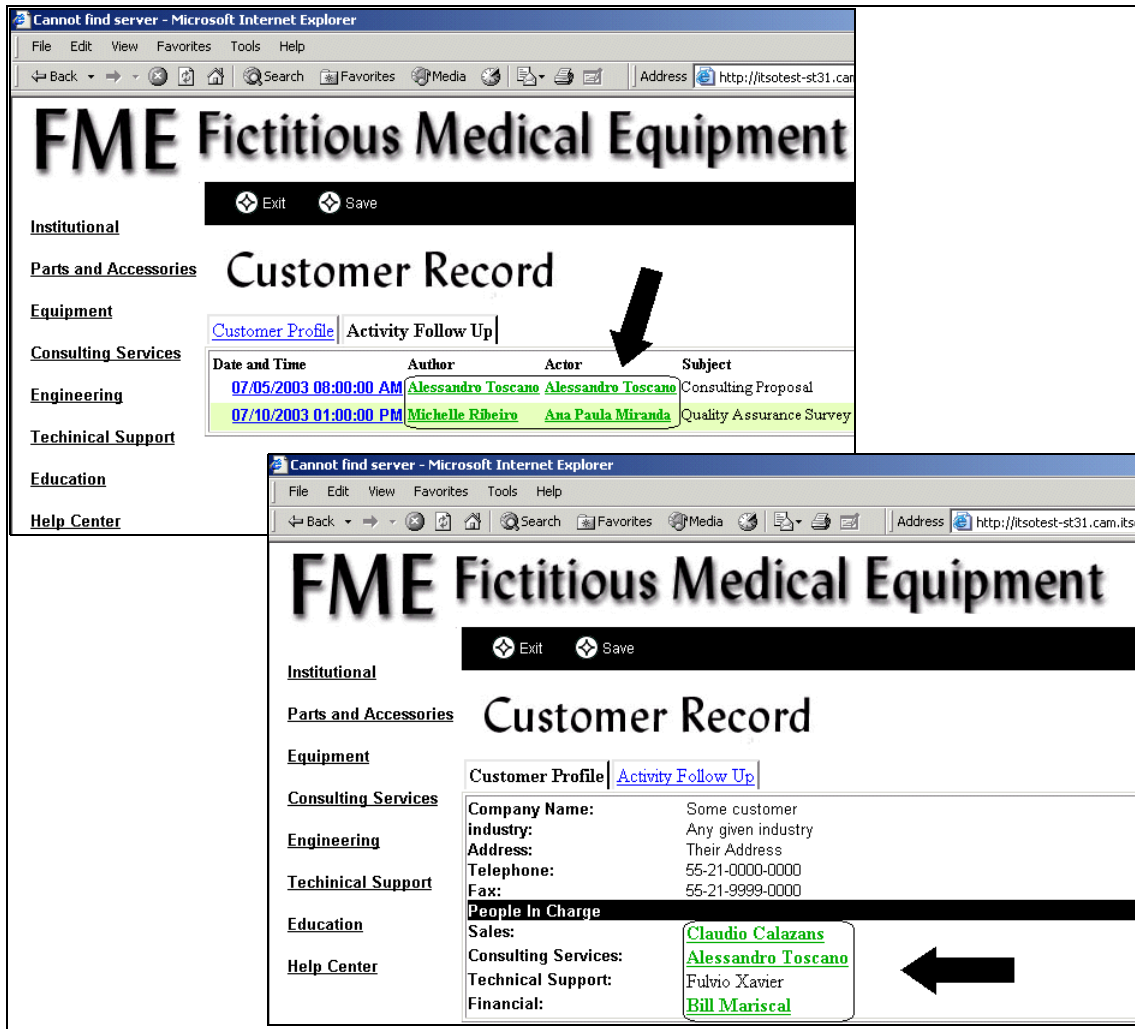


Figure 13-9 Live names within the customer tracking application

13.6.2 Allow offline message delivery

The next business requirement for Fictitious Medical Equipment is to be able to send offline messages to users not currently logged in to Sametime. Messages sent using this method will appear in a new Sametime window to the intended recipient as soon as they log in.

Why is this feature useful, since it is possible to simply send an e-mail instead? This feature can prove extremely useful for the following reasons:

- ▶ After discovering that a user is offline, based on their status within the Sametime buddylist, it is possible to still send a message *while remaining within the context of the Sametime buddylist*.
- ▶ As soon as the intended person changes their status to online, he or she will immediately receive the message.
- ▶ When sending an e-mail, it can take several days to get a response from the intended person.
- ▶ This same feature can be extended to work with announcement engines within the company later.

Methodology for implementing offline message delivery

A variety of techniques can be used to accomplish this functionality. Which approach to choose ultimately depends on your particular needs and willingness to consider deploying a customized Sametime client. The possible approaches are discussed in greater detail below.

Basically, we need two components to deliver this solution. The first component is a server application that handles the Sametime storage service in order to store the offline messages and also handle the users' login events.

The second component involves the Sametime client. This is where several different approaches to the solution are available:

- ▶ Possible Approach 1: Use a proprietary, custom developed Sametime client that identifies offline users on the buddylist and allows the user to send messages to the server application. (The custom developed client can be based on Java, COM, or C++, depending on your needs.) Note, however, that this option may require the user to deal with two different Sametime clients, namely the Sametime Connect client and the proprietary, custom developed client. Also, this solution may require an electronic distribution method for deploying the customized client to a large number of machines. For a large scale deployment, this could be a major effort.
- ▶ Possible Approach 2: Customize the Sametime Connect client or the Sametime Java Connect client in order to allow the same behavior described in the previous bullet. This is a more convenient solution in that it eliminates the need for two Sametime clients, but it requires privileged and very limited access to the source code of the Sametime Connect client. It also requires a software distribution solution.
- ▶ Possible Approach 3: Use a bot to send the offline message to the server application. This is the easiest solution from a development and administration perspective. Accordingly, it is the preferred approach. The user

must simply call the offline messenger bot and enter the user's name and message to be delivered. The only minor restriction is that the information must be entered using a specific syntax.

Based on the current scenario, we decided to implement the bot solution. If you wish to learn more about how to implement one of the other two options suggested above, please refer to the section “Offline Messenger Sample” in the *Sametime 3.1 Community Server Toolkit Tutorial*. Details are provided in “Part 1: Implementing the offline messenger server application” on page 385 for finding this tutorial.

Figure 13-10 illustrates the relationship between the portions of this solution.

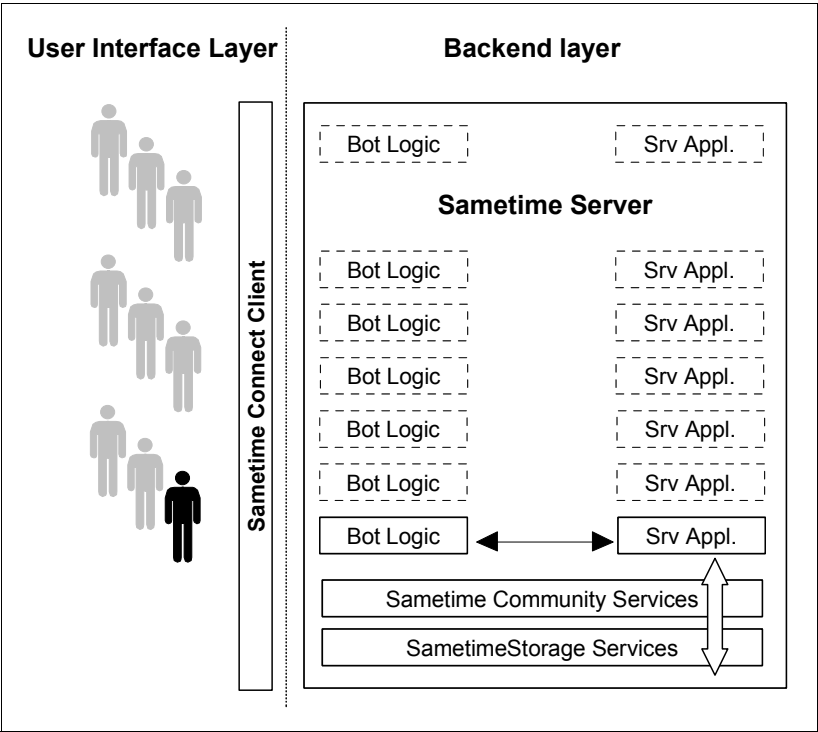


Figure 13-10 Offline messenger portions relationship

Part 1: Implementing the offline messenger server application

For this solution, the server side portion is implemented using the OffLine Messages sample, available within the Sametime Community Server Toolkit samples. This toolkit tutorial can be downloaded from a Sametime server that has the Sametime Software Development Kit (SDK) installed. You should be

able to access a listing of the toolkits on your Sametime server using the following URL:

`http://<sametime server name>/sametime/toolkits/`

Figure 13-11 illustrates a listing of the toolkit links available from the SDK page of a Sametime server with the toolkits installed.

Lotus. Sametime

The Sametime Software Development Kit (SDK)

Here you can download toolkit APIs, documentation, and samples in order to start adding Sametime capabilities to your applications. The following Toolkits are currently available:

Sametime 3.1 Toolkits	Description
Java Toolkit	Provides Sametime Community and Meeting services, allowing Java developers to embed real-time collaboration technology into new or existing applets, applications, and web sites
C++ Toolkit	Provides a collection of static library components that allow developers to enhance Windows applications with Community Services, such as awareness and instant messaging
COM Toolkit	Provides a COM object with basic Sametime Community Services (login, awareness, and instant messaging) for Visual Developers
Sametime Links Toolkit	A HTML/JavaScript API for adding "live names" to web applications or web pages with a few lines of HTML
Community Server Toolkit	Server API to develop server applications, get events, change attributes, and administer places
Directory and Database Access Toolkit	C APIs to modify Sametime for existing directories and databases
Chat Logging API	Allows the administrator to determine where and how logged chats are stored, by developing a DLL (or shared library) that

Figure 13-11 Listing of the toolkit links from a Sametime Server with the SDK installed

Once you click on the **Community Server Toolkit** link, you will see the tutorial material available for download available in PDF format. This is shown in Figure 13-12.

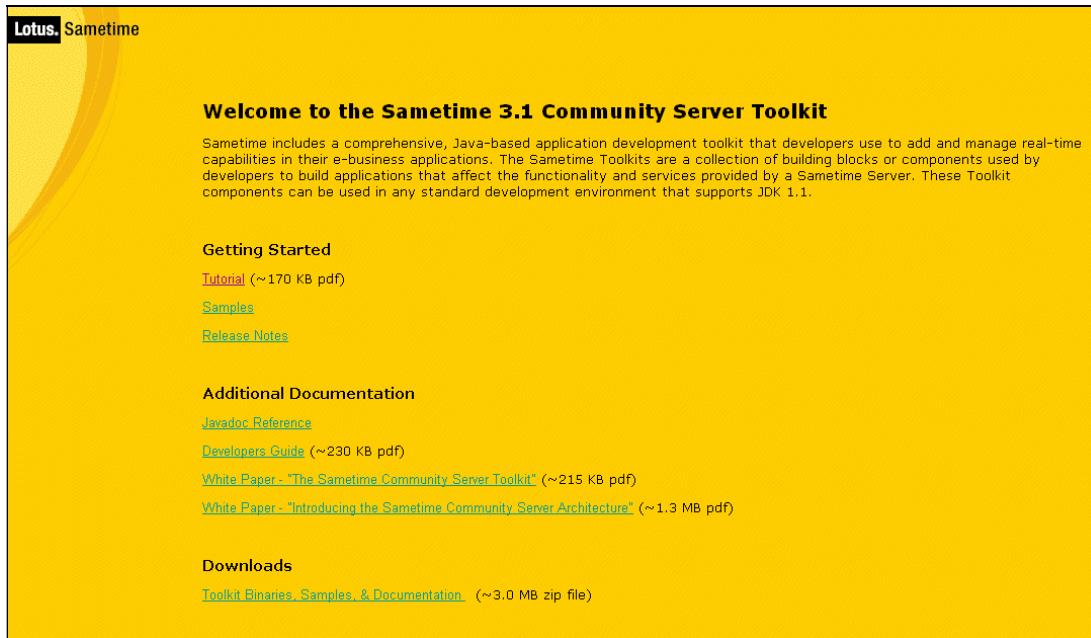


Figure 13-12 Tutorial for the Sametime Community Server Toolkit

Once you have downloaded the tutorial chapter, please refer to Chapter 3, "Offline Messages Sample" for detailed technical information on how to implement the Offline Messages Server Application used in this solution.

Note: The Sametime 3.1 Community Server Toolkit is available once the Sametime Toolkits have been installed on the Sametime server. As an alternative to following the links shown in the preceding figures, check under the html directory for a subdirectory named sametime/toolkits/st31commsrvtk/doc for the documentation mentioned above.

Part 2: Implementing the offline messenger bot

To implement the client side of this solution, we develop a custom bot. (For more detailed information on bots, please refer to Chapter 3, "Sametime Bots" on page 49). This bot is intended to send the offline message to the server application.

To implement this client portion of this solution, we use two techniques. The bot component is discussed in this redbook. The other technique refers to a portion of the code available in the `Client.java` file, available in Chapter 3, “Offline Messages Sample”, in the *Sametime 3.1 Community Server Toolkit Tutorial*.

Important: This section is intended to illustrate bot concepts at a higher level, without providing detailed techniques on bot development and implementation. We are just showing how previous discussed techniques can be modified to meet a specific requirement. Accordingly, the code shown here is for illustrative purposes.

For specific technical information on bots, their structure, and how to implement them, please refer to Chapter 3, “Sametime Bots” on page 49.

Basically, we are implementing a regular bot with a hook to a server application. To accomplish this task, we do the following:

- As in any bot, we implement a welcome message when the user opens the instant messaging session with the bot. To accomplish this, we put some code on the `ImReceived` event listener, as shown in Example 13-4.

Example 13-4 Basic ImReceived event response to users

```
public void imReceived(ImEvent e) {
    e.getIm().addImListener(this);
    e.getIm().sendText(true, "Hello " +
        e.getIm().getPartner().getName());
    e.getIm().sendText(true, "Welcome. I am the Offline
        Messenger Bot");
    e.getIm().sendText(true, "Please enter <user ID> #
        <message to be delivered>");
}
```

The code shown in Example 13-4 gives the user the result shown in Figure 13-13 on page 389.

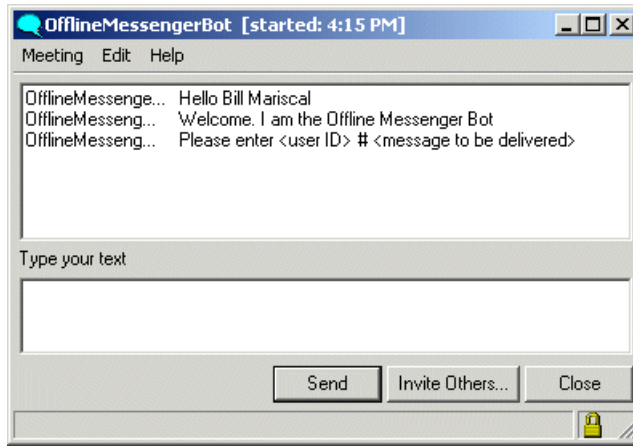


Figure 13-13 Offline Messenger Bot welcome message

- Next, we implement `sendOfflineMessage` (see Example 13-5). This code is responsible for sending the server application the message and the user name of the target offline user.

Example 13-5 `sendOfflineMessage` class

```
void sendOfflineMessage(STUser user, String message) {
    NdrOutputStream outStream = new NdrOutputStream();
    try
    {
        user.getId().dump(outStream);
        outStream.writeUTF(user.getName());
        outStream.writeUTF(message);
    }
    catch(IOException e)
    {
        e.printStackTrace();
        return;
    }
    Channel channel =
        m_channelService.createChannel(
            SERVICE_TYPE, 0, 0,
            EncLevel.ENC_LEVEL_ALL,
            outStream.toByteArray(), null);
    channel.open();
}
```

- Finally, we handle the message received from the user in order to resolve the target user name and the message to be sent.

When a user sends a message to the bot, the `textReceived()` event is triggered.

As the server application and the code to create the communication channel with this server application is ready, all we need to do is treat the text received on the event `textReceived()` and then invoke the `sendOfflineMessage()` code, passing the target user as a `STUser` object and the message to be delivered as a `String` object.

13.6.3 Implement workflow using Sametime awareness feature

The Offline Messenger Bot described in the previous section is only one way to implement a solution for dealing with offline users. In some cases, it may be desirable to integrate Sametime instant messaging functionality into a more complex application that already has existing workflow logic built in. When integrating Sametime, the workflow application needs to determine a user's online status and then proceed to either send the message via Sametime or e-mail, depending upon the person's availability.

This concept can be illustrated within the context of FME's incident tracking application. This application already has workflow logic based on support escalation business rules in order to notify specific people and departments of software and equipment problem reports.

Figure 13-14 on page 391 gives an overview of the FME's support escalation process.

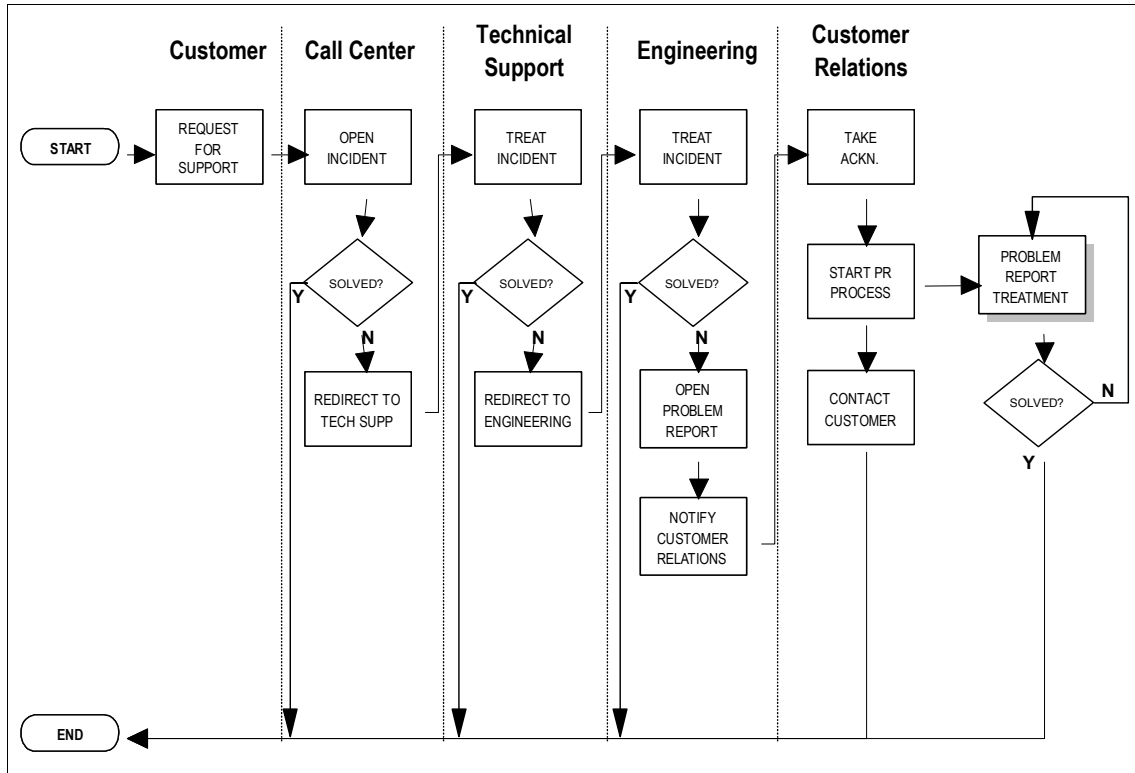


Figure 13-14 Support escalation process overview

This process is managed by the Incident Tracking application, which is currently a Lotus Notes/Domino application. However, people at FME want to implement Sametime features within this process to enable instant notifications to the person in the next step of the workflow cycle.

Finally, since there is a dedicated chapter to Sametime and Workflow in the redbook, this section will not illustrate the detailed implementation steps for integrating Sametime into a workflow application. The user should refer to Chapter 6, “Sametime and workflow” on page 139 for more detailed information. Instead, this section is intended to provide an overview of the available approaches and benefits within the context of FME’s business scenario.

Methodology for integrating Sametime into workflow

The workflow process is already managed by the Domino based incident tracking application. While integrating Sametime would not alter the basic workflow process, the application would need to be modified to first check for a user’s online status within the next workflow step. Based on the status, the

application can send an e-mail or an instant message to the user, notifying them about the problem report document.

To implement this status checking feature within a workflow application, we can choose from at least two available techniques:

- ▶ This feature could be implemented by using the Java Toolkit, essentially creating a Java agent that uses the Sametime Java Toolkit features to check for user availability. For a complete example on how to implement status checking functionality using this method, please refer to 6.3, “The AnnouncementSender application” on page 141. There you will find step by step instructions on how to implement a Sametime enabled workflow application.

Note: While we are using a Notes application for this specific example, the use of this technique is not by any means restricted to Notes based applications only. The techniques described at 6.3, “The AnnouncementSender application” on page 141 are Java compliant methods. Accordingly, this approach can be used in any Java based application environment.

- ▶ The second approach to checking a user’s online status is to use a Web service. From an architectural perspective, this approach has significant advantage. Using a Web service, the logic of querying the user status within the Sametime environment can be re-utilized by any application within the enterprise.

For a complete example on how to implement an workflow application using a Web service, please refer to 6.4, “The AnnouncementSender Web service” on page 148. There you find step-by-step instructions on how to implement a Sametime-enabled workflow application.

13.7 Phase 3: Expanding to outside world

With Sametime features working to help increase productivity for FME employees, FME now wishes to shift its focus and expand these tools to its outside customer base. Using Sametime Toolkits, we extend the functionality to create a more direct and efficient link between the external customers and the internal employees.

The following upcoming sections discuss how FME would implement new functionality to help its customers:

- ▶ Section 13.7.1, “Provide users with self service tools” on page 393

- ▶ Section 13.7.2, “Provide customers ability to call for online support” on page 400
- ▶ Section 13.7.3, “Track user activity and provide active call center behavior” on page 408
- ▶ Section 13.7.4, “Maintain logging of online customer conversations” on page 411
- ▶ Section 13.7.5, “Allow educational sessions with customers” on page 413
- ▶ Section 13.7.6, “Implementing multilanguage educational sessions” on page 417

Important: Please be aware that when we talk about expanding Sametime features outside of the company’s network to an extranet, some serious deployment considerations need to be addressed. For example, there are specific firewall and reverse proxy solutions that need to be designed, implemented, and configured very carefully. The objective of this chapter and of this redbook is not to deal specifically with the deployment related issues. As a deployment reference, we recommend that you review the following sources:

- ▶ *Sametime 3.1 Installation Guide* (see 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22 for details on how to find this publication).
- ▶ *Sametime 3.1 Administration Guide* (see 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22 for details on how to find this publication).
- ▶ *Lotus Sametime 2.0 Deployment Guide*, SG24-6206. This redbook is based on a old version of Sametime. However, it gives a good understanding of basic Sametime architecture and deployment concepts.

13.7.1 Provide users with self service tools

FME has a high volume of repetitive calls that concern the following questions:

- ▶ Estimated delivery dates for equipment and sales proposals
- ▶ Checking the status of a support incident
- ▶ Checking the status for a custom equipment order
- ▶ Checking delivery status for medical apparel

If the mechanism to provide a response for these common questions can be properly automated through the company’s Web site, this would reduce the load on the call center attendants and provide faster customer response time.

FME has decided to try and implement tools to provide the following functionality:

- ▶ Provide customers with a self service tool that allows them to check for estimated delivery dates for equipment and sales proposals
- ▶ Provide customers the ability to view the latest status for support incidents
- ▶ Provide customers the ability to track logistics and shipping information
- ▶ Provide customers the ability to check on the status of custom orders

Methodology for enabling self-service tools

In order to enable a base level of self-service functionality from FME's external Web site, several technical approaches are possible:

- ▶ Develop a unique interface that can access data from internal applications
- ▶ Duplicate (replicate) a data set of required information on a scheduled basis
- ▶ Extend each unit application's interface to the external Internet Web site

While each of the above options seems practical, each approach would require significant architectural planning and infrastructure design, as well as a large degree of custom development.

Sametime can give us much of the required functionality through bots. Bots can be thought of as automated machines or "agents" residing on your Sametime server, and accessible from within your Sametime buddylist. They provide a response to questions posted to them. By using the bot approach, FME can satisfy the requirement for external self service tools quickly, securely, and with minimum development effort relative to the approaches previously discussed in this section. For detailed information about bots and how to develop and deploy them, please refer to Chapter 3, "Sametime Bots" on page 49.

For illustrative purposes, we will focus first on a support "Incident Tracking" self-service tool. This application allows a customer to enter an incident tracking number and then see the latest status of the issue.

Figure 13-15 on page 395 shows the logic flow of the bot that accomplish this request.

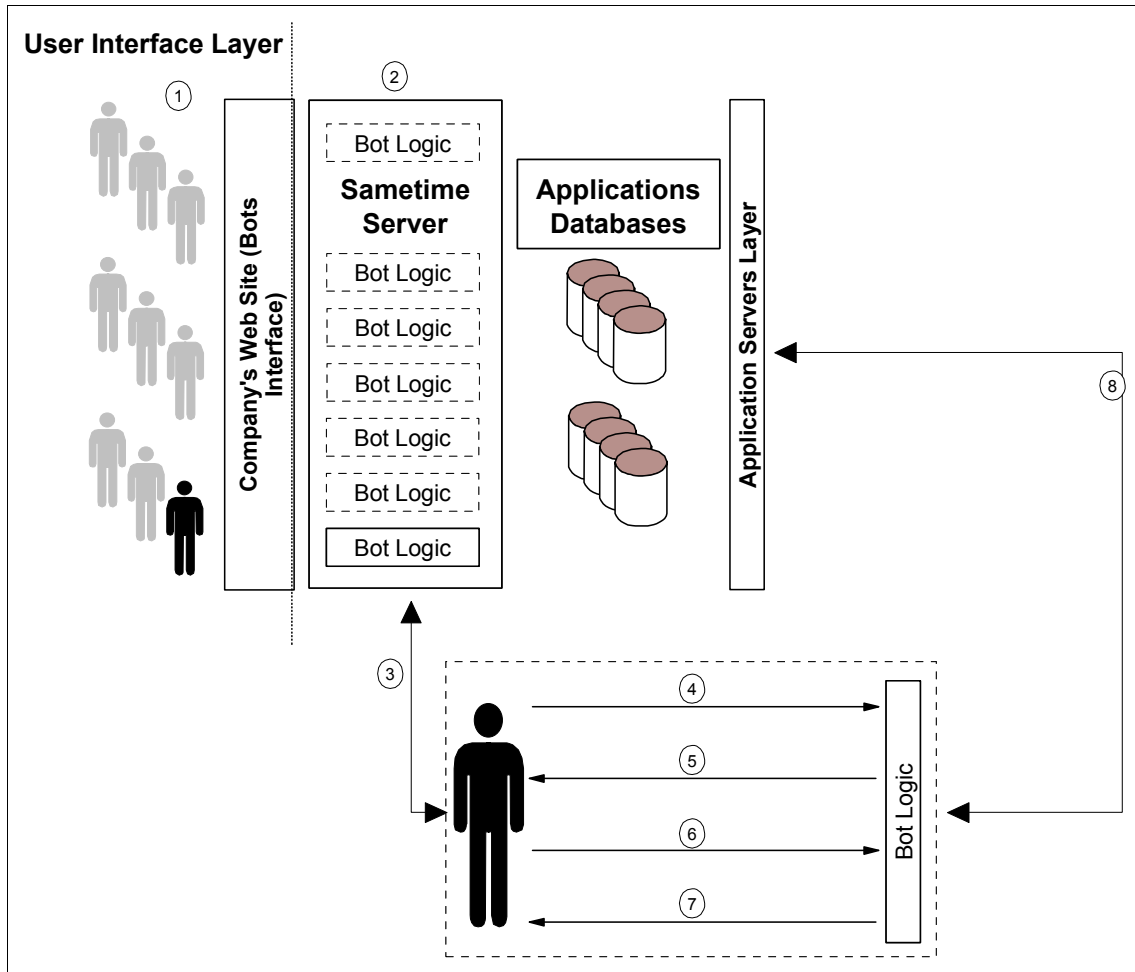


Figure 13-15 Logic flow of the Incident Status Tracking Bot

Now we explain the logic presented in Figure 13-15 step-by-step:

1. At the user interface layer (FME's Web site), customers can choose which self-service tool they want to use. Each tool is attached to an internal Sametime Bot.
2. The bots reside on the internal Sametime server. Each bot contains its own logic and functionality.
3. By choosing a bot through the interface layer (Web site), the customer is automatically connected to this bot.
4. Without further customer intervention, an event is sent to the bot, indicating that someone wants to chat.

5. A standard welcome message is sent to the customer informing them how to use the specific bot. If a specific syntax is required, this is indicated through the bot interface window.
6. The customer then sends his query to the bot, which in turn interprets the query.
7. At this step, the bot calls the proper internal application, which searches for the information customer requested in the back-end store. (In the case of this example, we used a Notes database, and used the native Java classes provided by IBM Lotus Notes and Domino.)
8. Finally, the bot returns the desired information to customer, based on the submitted query.

Important: In this specific example, we used a Notes database as the back-end data store. Since the bot architecture is Java compliant, however, we can use any back-end source that is also Java compliant. For example, you can use bots to directly access relational databases such as IBM DB/2, Oracle, MS SQL Server, Informix®, and so on. You can also use the application servers' built-in logic, such as with Siebel, SAP, PeopleSoft, and any other open standard application servers that supply a Java interface.

Implementing the bot

Since the subject of designing and implementing bots has already been well documented in Chapter 3, "Sametime Bots" on page 49, we will not repeat all of the implementation details here. Instead, our goal is to just provide the necessary logic to help you understand what the bot does and how it works at a high level.

Important: The incident tracking bot we are implementing now is completely based on the bot example available on the 3.3.2, "FAQ Bot" on page 61. Just minor changes must be made in order to transform the "FAQ Bot" into the "incident tracking" bot, using the FAQ Bot example.

After properly initializing the necessary services, logging in to the server, and receiving the incident number from the user, is time for the bot to give information back to the user. In order to access the incident tracking application and locate the desired incident, we used the "query" portion of the code from the FAQ Bot example to illustrate how to do this (Example 13-6 on page 397).

```
public String[] findIncident(String query)
{
    String[] strIncidentFw = "";
    Session session = null;
    try
    {
        //Start a notes session to access the Incident Tracking database
        NotesThread.sinitThread();
        session = NotesFactory.createSession("itsotest-st31.cam.itso.ibm.com");
        Database db = session.getDatabase("itsotest-st31/ITSOST31",
        "IncidentTracking.nsf");
        if (db == null)
        {
            System.out.println("Can't get a handle on the Database");
        }
        else
        {
            //Do a full text search on the database
            DocumentCollection dc = db.FTSearch("[IncidentNumber] Contains \"" +
            query + "\"");

            Document doc;

            if (dc.getCount() <= 0)
            {
                //If there are no matches...
                strIncidentFw =
                "Sorry I could not find your incident!";
            }
            else
            {
                //Return the answer
                doc = dc.getFirstDocument();
                strIncidentFw = doc.getItemValueString("Followup");
            }
        }
    }
    catch (NotesException en)
    {
        System.out.println(en.getMessage());
        en.printStackTrace();
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
    finally
    {

```

```
        try
        {
            if (session != null)
            {
                session.recycle();
            }
        }
        catch (Exception x)
        {
            x.printStackTrace();
        }
        NotesThread.stermThread();
    }
    return strIncidentFw;
}
```

The results

As users enter the tracking incident number, they are provided with updated information concerning the status and progress of resolving the incident.

Figure 13-16 on page 399 illustrates an example of a customer to bot conversation.

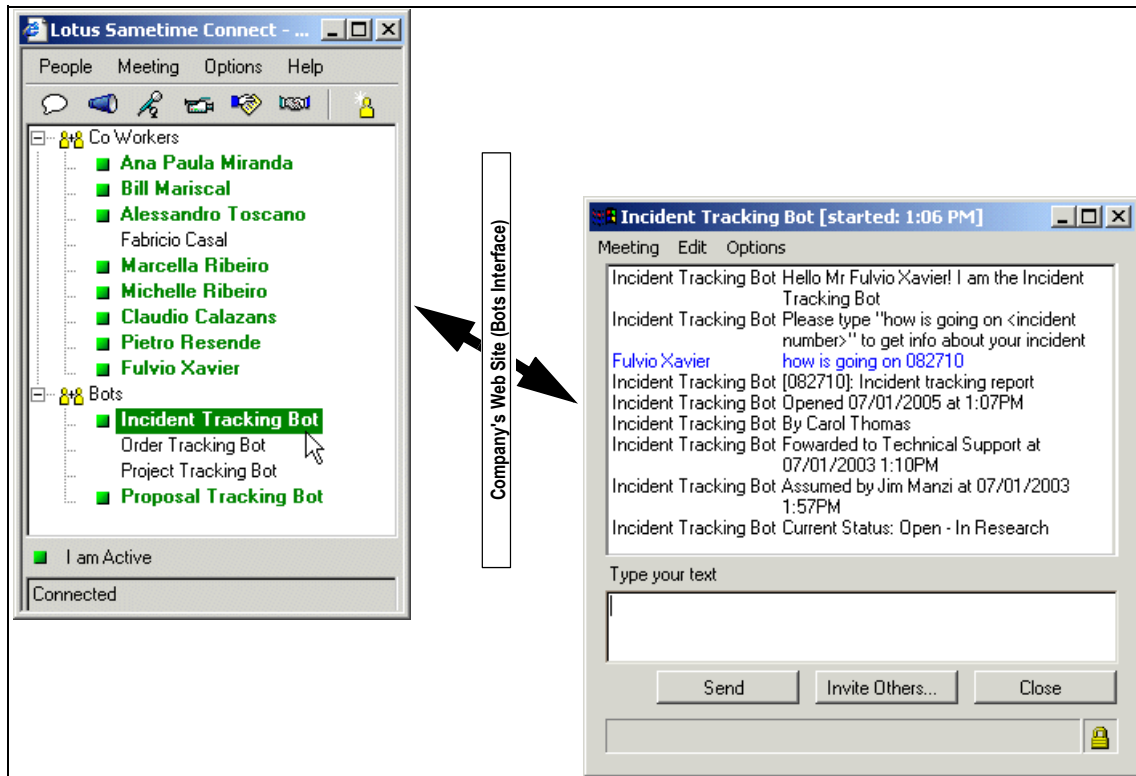


Figure 13-16 Customer to bot conversation on incident tracking application

Self-service tools and the customer relationship model

One of the most important changes to result from the implementation of self-service tools involves the customer relations model originally presented in Figure 13-1 on page 363. By providing the customer with more tools and an increased ability to help themselves, this changes the dynamics of communication. No longer is it necessary for all communication to route through the call center.

In Figure 13-17 on page 400, the dotted arrows illustrate the new communications channels created by this implementation.

Attention: As we continue through this chapter, you will begin to see a fundamental shift in the customer relations model. Communication will no longer only flow through the call center interface. The goal is to make it a much more collaborative model.

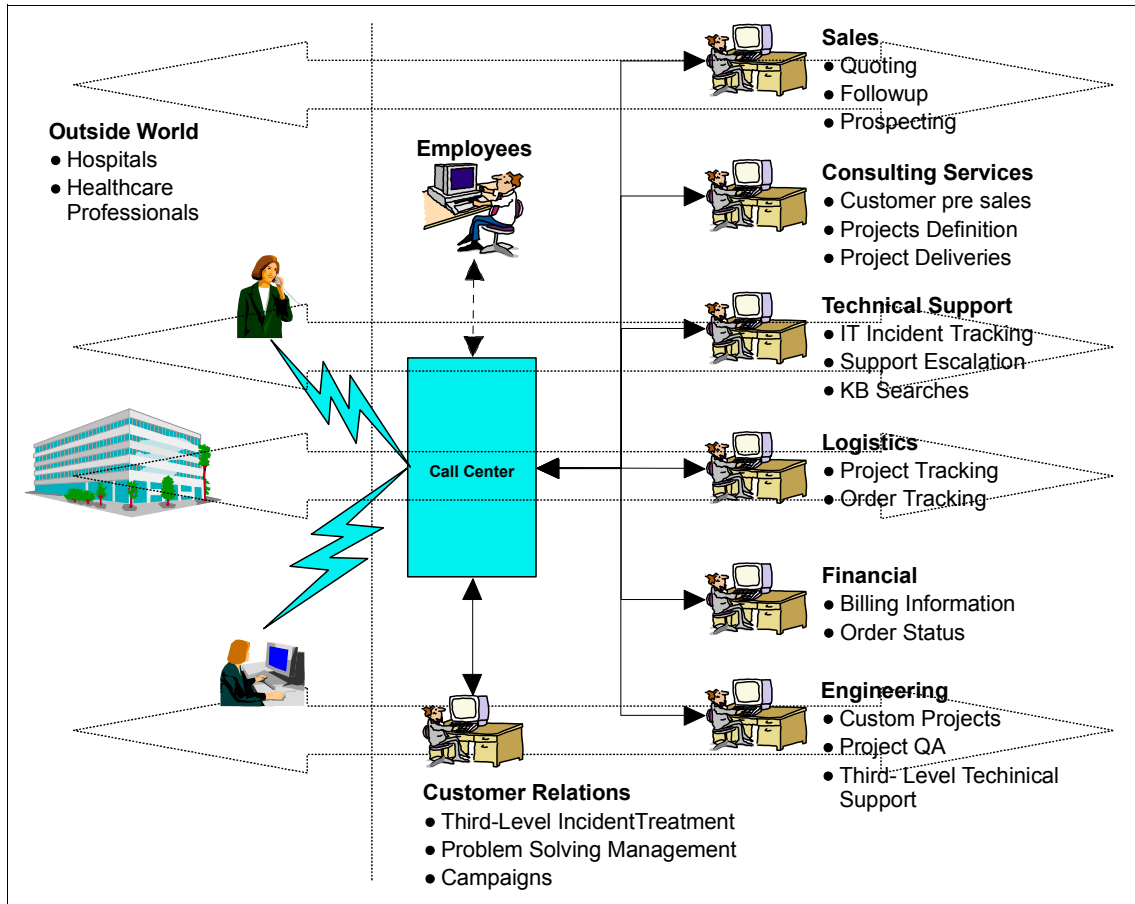


Figure 13-17 New communications channels made available

13.7.2 Provide customers ability to call for online support

After implementing the solution described in 13.7.1, “Provide users with self service tools” on page 393, Fictitious has decided to continue enhancing the capabilities provided through their Web site.

If customers can communicate with bots, why not go one step further and provide the capability for customers to interact and chat directly customer service representatives and call center attendants? FME’s requirement is to allow customers to click on a link within their Web site and have this bring up a chat window, enabling real-time dialog with a company representative.

Defining the business rules for online customer dialog

Prior to implementing the technical aspects of this scenario, there are specific business rules that need to be defined. These will help further define the requirements of the technical solution.

- ▶ Call center attendants will not be dedicated to online dialog. Instead, call center attendants must balance their efforts between telephone conversations and online requests via Sametime.
- ▶ Chat dialog windows must be limited to one window per session. Too many windows with customer questions would present a significant distraction to the attendant. Once an attendant is interacting with a customer, they should be marked as “unavailable” until their dialog is completed and they are ready for the next question.
- ▶ Since the call center is also responsible for internal employees calls, a mechanism must be in place to give priority to outside customers.
- ▶ A customer call cannot be directly addressed to a specific attendant. (This is at least a limitation for the initial rollout.)
- ▶ The call center attendant controls their available status, and must have the option to attend the next customer at their own pace.

Based on these business rules, the following features will be required:

- ▶ A queueing mechanism must be implemented to prioritize and manage multiple users’ requests.
- ▶ This queue must be available for all attendants to see.
- ▶ Each attendant must be able to identify if another attendant is already chatting with a customer.
- ▶ There must be at least two separate queues for questions. One queue will be for external customers, while the other will serve internal employees.

Further visualizing the solution

Finally, it helps to use a commonly known analogy to better conceptualize the scenario/solution we are trying to build. Perhaps the best analogy is one of bank tellers serving customers at a retail banking branch. Here are some characteristics that apply to both the retail banking teller scenarios, as well as to our proposed approach to the online support interface.

- ▶ The teller window is composed of “n” tellers, each one with similar responsibilities, and essentially performing the same job. If one teller steps away, another can substitute for them.
- ▶ There is usually a single line of customers, waiting to be served by the “n” tellers available.

- After each teller completes their task helping a customer, they signal to the next customer in line that they are now available.

The technical approach

In building this solution, we will leverage capabilities provided by the Sametime Links Toolkit. We will also work with concepts based upon Places and Place Architecture.

As a prerequisite for understanding the upcoming technical approach, we strongly recommend that you review both Chapter 9, “Sametime Links” on page 219 and Chapter 8, “Places and Place awareness” on page 183.

Additionally, you may find it helpful to review Chapter 12, “The Sametime Links Toolkit”, from the IBM Redbook *Working with the Sametime Client Toolkits*, SG24-6666. This chapter discusses the Sametime Links Toolkit and how to create Place-based awareness through the Sametime Links Toolkit.

Building the solution

Figure 13-18 on page 403 illustrates the solution we are trying to build. It introduces how we will integrate the concept of Places and Place architecture into the solution.

- By clicking on a link at the Web site, the customers gain access to the queue for the Call Center.
- Call center attendants have the ability to see who has entered the Place.

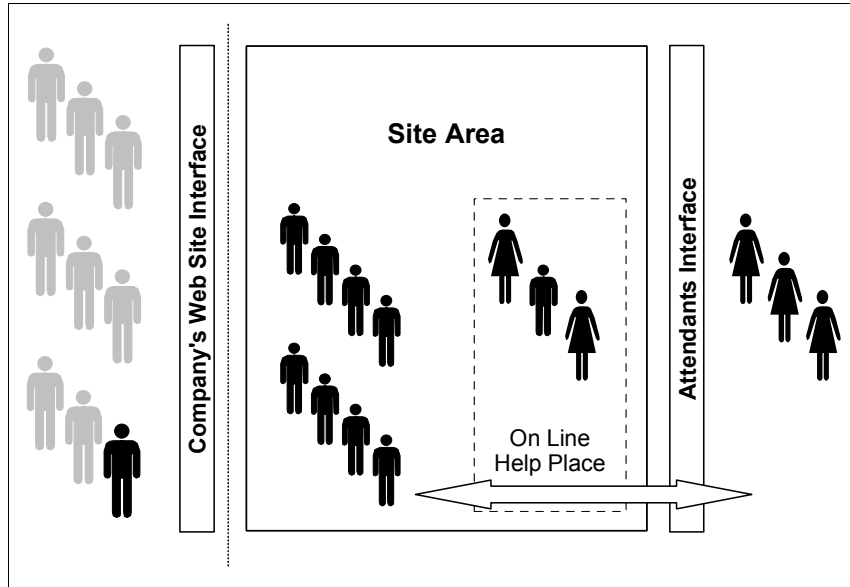


Figure 13-18 Call center place architecture

The description of how to build the solution will be discussed in discreet parts.

Step 1: Implementing the customer interface

In order to implement the customer part of this scenario, we apply similar techniques to those described in 13.6.1, “Enabling people awareness into existing applications” on page 379. In this case, however, we use a few more resources from the Sametime Links Toolkit.

When the user clicks on a link, he enters the Online Help Place. To do this, we must call the following functions:

- Use the `STLinksEnterPlace()` function to put the user into the `OLSupport` Place. This Place will hold each customer request to access online support. To do this, we implement a JavaScript function to be called when the requested link is pressed. This is shown in Example 13-7.

Example 13-7 JavaScript function to make the user enter a Place

```
function callSupport()
{
    STLinksEnterPlace("OLSupport", true );
    window.open("<welcome form url>", "", "height=200,width=300")
}
```

Attention: The <welcome form url> stands for the form used to tell the customer that he is on a queue. For each environment, a different approach can be implemented.

This code also assumes that the writeSTLinksApplet() function has been called and the user is properly authenticated on the server as a registered user or as anonymous

Executing the code shown in Example 13-7 on page 403 within our implementation give us the result shown in Figure 13-19.



Figure 13-19 Screen example when the customer is queued

The resulting customer interface

From a customer perspective, customers will click a link requesting help from the FME Web site and will be presented with a screen acknowledging their request and asking them to please wait for the next available attendant. Once the agent becomes available for real-time dialog, a separate chat window will be opened.

Figure 13-20 on page 405 illustrates the customer user interface with a dialog taking place.

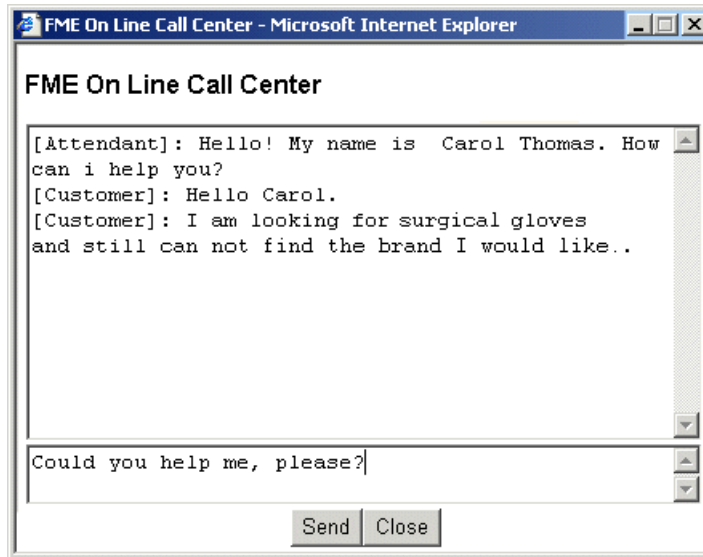


Figure 13-20 Customer side user interface for online help

Step 2: Implementing the online attendant interface

The attendant interface prompts the attendant for authentication, then prompts the attendant to enter into the OLSupport Place. Once there, attendants are able to see who is present within the Place and which Place chats are underway.

Once entering the support Place, the attendant will see text indicating that there are customers in the queue. By clicking on the link, an interface appears showing the attendant which customers are present.

To implement the functionality for attendants to log in and have a Place where they can manage users in a queue, we must implement the following steps:

1. First, we added the code shown in Example 13-8 into the HTML HEAD tag of the customer record form to set the necessary parameters to call Sametime Links.

Example 13-8 Enabling the attendant interface for online help

```
<LINK REL=STYLESHEET HREF="http://<st server>/sametime/stlinks/stlinks.css"
TYPE="text/css">
```

```
<SCRIPT src="http://<st server>/sametime/stlinks/stlinks.js"></SCRIPT>
```

```
<SCRIPT>
  setSTLinksURL("http://<st server>/sametime/stlinks","en",
  "http://<st server>/sametime/stlinks")
```

```
</script>
```

2. Next, we call an authentication interface to get the user name and password of the attendant and log them into the Sametime server. This is shown in Example 13-9.

Example 13-9 Logging in the attendant

```
<script language="JavaScript1.2">
  var STUserName, STPassword;
  STUserName = prompt("Enter Sametime Login Name.", "");
  STPassword = prompt("Enter Sametime Password.", "");
  writeSTLinksApplet(STUserName, STPassword, false);
</script>
```

3. Next, call the function to have the Sametime Links Toolkit write the number of users on queue by using the writePlaceCounter() function, as shown in Example 13-10.

Example 13-10 Using the writePlaceCounter() function

```
<script>
  writePlaceCounter("OLSupport", "OLSupport", false );
</script>
```

4. Finally, provide a button for the attendant to see the list of customers currently waiting in the queue. We will do this using the openPlaceWin() function, as shown in Example 13-11.

Example 13-11 Using the openPlaceWin() function

```
<script>
  openPlaceWin("OLSupport", "Customer Support Queue");
</script>
```

The resulting attendant interface

Based on the code implemented in “Step 2: Implementing the online attendant interface” on page 405, Figure 13-21 on page 407 illustrates the potential UI for the attendant as they log into the system.

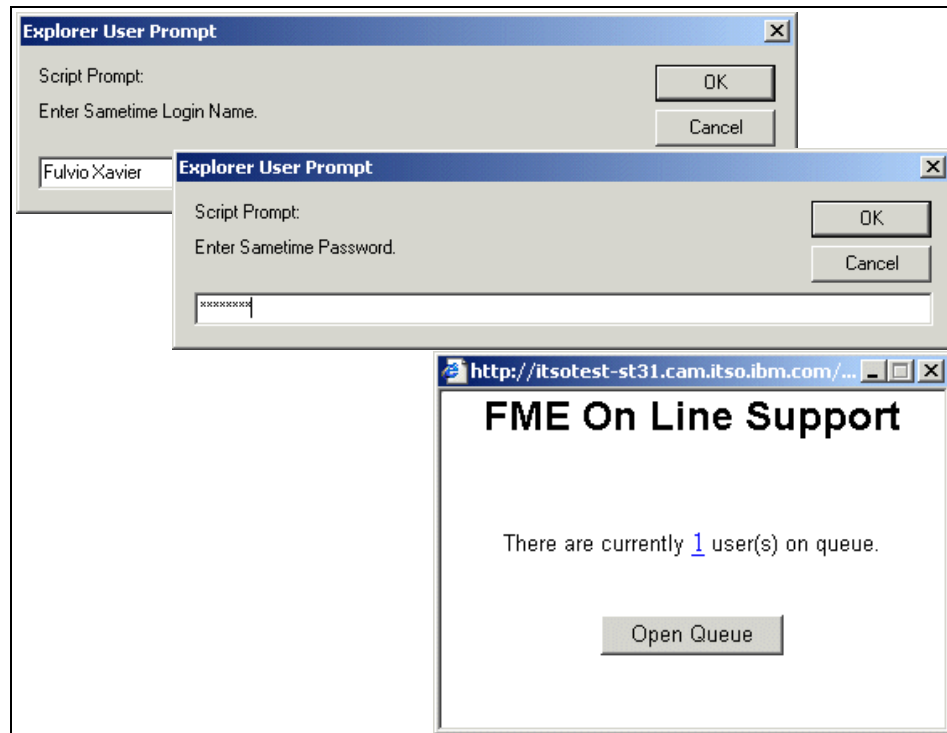


Figure 13-21 User Interface example for the attendant

Important: Please keep in mind that the solution described here (creating an online help desk) can be accomplished in several different ways. Using the Sametime Links Toolkit is one possible approach, which we considered to be the fastest approach with the least amount of effort. Please note, however, that it is possible to implement a full client-server solution based on the Java, C++, or COM Toolkits.

In Chapter 8, “Places and Place awareness” on page 183, the application example illustrated can be applied to address the needs of a call center application. It provides more advanced functions such as sections, server applications, and activities.

The following publications also show specific examples of how to create a call center solution, as well as other potential uses for Places and sections within a Place-based Sametime solution. These solutions have primarily been developed using techniques other than Sametime Links:

- ▶ Chapter 8, ““Sametime development”, of the *B2B Collaborative Commerce with Sametime, QuickPlace, and WebSphere Commerce Suite*, SG24-6218
- ▶ *Working with the Sametime Client Toolkits*, SG24-6666
- ▶ *Working With Sametime Community Server Toolkit*, SG24-6667

13.7.3 Track user activity and provide active call center behavior

Now we have a powerful communication tool available for customers to chat with call center attendants. FME wishes to implement the capability to monitor which areas of the Web site customers are visiting most frequently. Ultimately, this capability will allow FME to better understand their customers’ needs and hopefully improve service. If call center attendants know which page or area of the Web site a customer is currently on, then the attendant can proactively offer assistance.

How do we accomplish this request?

To keep track of who is on the Web site we will leverage features of Sametime Links and make use of Sametime’s ability to accept both anonymous logins, as well as logins for registered users. We will also make use of its ability to track Places. For a detailed review of how to implement this capability, please refer to “Allow an agent to see who is on their Web site” on page 247.

In this case, the virtual Places are sections of the Web site, such as the help center, or any of the pages focused on informing the customer about FME, monitoring systems, diagnostic images, parts and accessories, and so on.

These Places can be structured and defined in order to give the client side the ability to log into each Place.

We can also set attributes to users and send these attributes to the call center attendant application (previously mentioned in 13.7.1, “Provide users with self service tools” on page 393). The agent will know which page the customer is on, since we will set the attributes as the page where users are.

Figure 13-22 illustrates the user flow within the Web site, based on Places and Places awareness context.

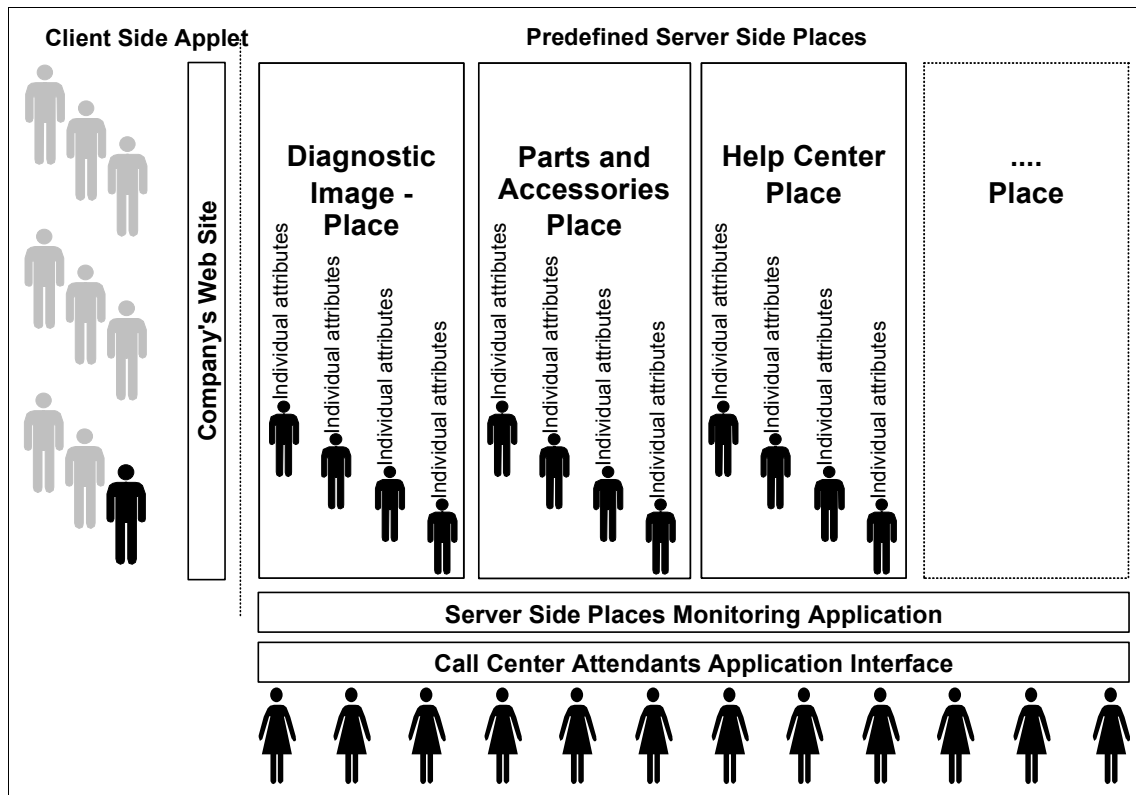


Figure 13-22 User flow within the Web site

Also, the call center agent client must be adjusted to accomplish this request. Now the attendants must have some space on their client to “see” where customers are.

Implementing the customer side

Within this section, we illustrate how we implement the customer side of the requested feature.

In order to accomplish this task, we will use the Sametime Links Toolkit. Keep in mind, however, that we cannot directly set/update the user properties with information about the exact page the user is navigating to. In order to handle this, we must implement a type of workaround. We can set the information we need at the user status message property and then manipulate the text stream of the URL.

Following the techniques used in the 13.7.1, “Provide users with self service tools” on page 393, we now need to set the customer status message on every page visited. In order to provide this functionality, we must use the `STLinksetMyStatus()` function, as shown in Example 13-12.

Example 13-12 Setting the user on line status to show the current page

```
<script>
function STLinksLoggedIn() {
    STLinksSetMyStatus("32", "On Page: " + window.location.pathname);
    STLinksEnterPlace("WEBSITE", true)
}
</script>
```

This example shows that when Sametime Links receives the `STLinksLoggedIn` event, the user status message is changed to show the page where the user is positioned on the Web site. After this, we enter into the "WEBSITE" Place.

Note that the `OLSupport Place` is used for users requesting online support and the "WEBSITE" Place is intended as the generic Place for the whole Web site. Also note that according to Figure 13-22 on page 409, predefined Places can be defined to track users on different areas of the Web site. In this case, we would use some attribute on the site pages to fill the parameter currently receiving the value "WEBSITE".

Important: As mentioned before, the techniques used here are the same as the ones used in 13.7.1, “Provide users with self service tools” on page 393 and in 13.6.1, “Enabling people awareness into existing applications” on page 379.

Please refer to Chapter 9, “Sametime Links” on page 219 and specifically 9.5, “Building an interactive Web site” on page 244 for detailed information about implementing this feature through Sametime Links.

Implementing the attendant side

Basically, the attendant side of this solution is done as an indirect result of implementing the customer side. Now, by just moving the mouse over the user name on the awareness list, the attendant is able to know which page the user is on.

The results

Figure 13-23 shows the attendant side of the application with the ability to identify which page the customer is on.

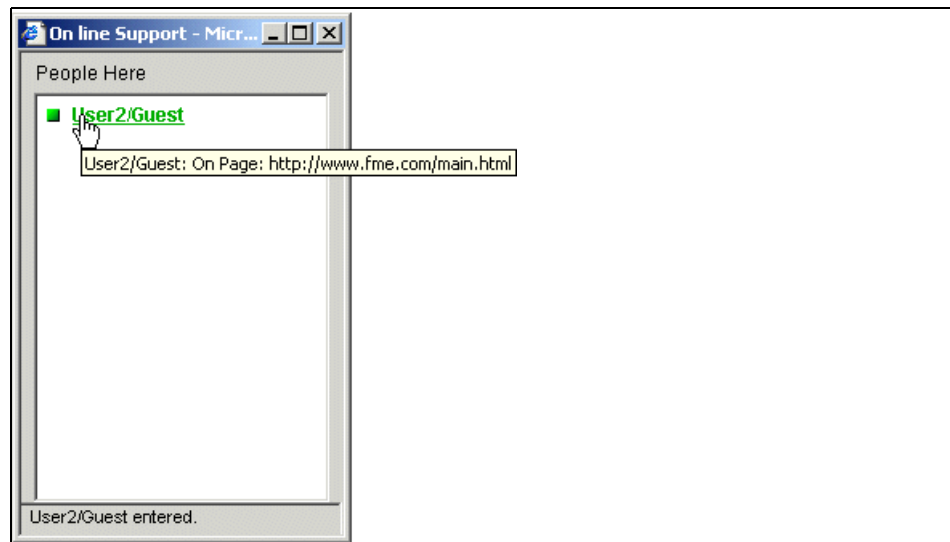


Figure 13-23 Attendant awareness application with current user page

From the user perspective, nothing changes. Sametime Links functionality is working in the background as the users navigates through the Web site.

13.7.4 Maintain logging of online customer conversations

So far, FME has implemented many features that let customers interact directly with agents inside FME. For auditing purposes and potential liability reasons, FME feels it would be beneficial to implement chat logging capabilities. The requirements and benefits of implementing this feature are reviewed in Chapter 5, "Chat Logging/DDA Toolkit" on page 127.

Implementing chat logging functionality

Since Version 3.0, Sametime implements a feature called Chat Logging Server Provider Interface (SPI). Using of the box features, with only slight modifications

to the template DLL provided in the SPI template, we can record every conversation taken within users on one or more Sametime servers. Note the following two additional requirements necessary to make this tool fully functional:

- ▶ There is no transcript retrieval tool available. We must design our own.
- ▶ The chat transcripts are recorded in text files located on a directory indicated on the `sametime.ini` file.

However, we can design our own DLL (W32) or shared library (AIX) based on a template offered by the SPI. In this case, we are able to address the characteristics listed above by recording the transcripts to a database, for example, then implementing some solution specific features.

Attention: For detailed information on the Chat Logging SPI, refer to the Chapter 5, “Chat Logging/DDA Toolkit” on page 127.

To implement Chat Logging within FME environment we must read and properly understand the Chat Logging characteristics and considerations. In order to understand the whole picture of the Chat Logging feature of Sametime, we refer to the *Sametime 3.1 Directory and Database Access Toolkit Developer's Guide* on the toolkit documentation. Also, we refer to Chapter 5, “Chat Logging/DDA Toolkit” on page 127.

Building and installing the chat logging DLL

It is best to follow these general steps for building and installing a chat logging DLL:

1. Study the provided sample, `StChatLogFile.dll`. To access the sample, click Chat Logging SPI Sample on the Directory and Database Access SPI Toolkit home page. For more information about the sample, refer to the “Chat Logging SPI Sample” section of the *Sametime 3.1 Directory and Database Access Toolkit Developer's Guide* documentation.
2. Use the template version `StChatLogDummy.dll` to create a new Chat Logging SPI implementation. To access the template version, click **Chat Logging SPI Header Files** on the “Directory and Database Access SPI Toolkit” home page.

Attention: When building your chat logging DLL, leave the sample unchanged.

3. When creating a new DLL, use the debug mechanisms and make sure debug messages are being saved in the trace files to facilitate troubleshooting during the development and use of the DLL.

4. Save the new DLL under the name StChatLog.dll.
5. Test the created DLL.

Attention: Do not install a new chat logging DLL until you have tested it thoroughly; DLLs directly affect server components operation.

6. Replace the dummy chat logging DLL on the Sametime server with the newly created DLL (StChatLog.dll). The new DLL must be placed in the default directory that contains all other DLLs. The default directory is C:\Sametime. If more than one server is included in the community, replace the DLL for each server.
7. Make the necessary changes in the stconfig.nsf file of each Sametime server in the community. For more information, see the “Administration” section in Chapter 2 of the *Sametime 3.1 Directory and Database Access Toolkit Developer's Guide*. This document is included as part of the Sametime SDK. For additional information on accessing this documentation, please refer to 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22.
8. To activate the created chat logging DLL, start and stop the Sametime server on which the DLL is installed. If the Sametime community contains more than one Sametime server, start and stop all servers in the community.

For more information on working with Sametime servers, refer to the *Sametime 3.1 Installation Guide* and the *Sametime 3.1 Administrator's Guide* (see 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22 for information on accessing these publications).

The results

By implementing this resource, we can record all chat dialogs placed between FME agents and external customers. Each chat is recorded on a separated text file beneath the directory specified on the Sametime configuration database.

13.7.5 Allow educational sessions with customers

Realizing the potential of the Sametime solution within the company, people at Fictitious Medical Equipment wish to continue extending the reach of the Sametime features to customers. As a final capability, they wish to provide functionality for online moderated discussions that can serve as educational sessions for customers.

The solution is based upon the following requirements:

- Customers should be able to connect through a browser based client.

- ▶ Anonymous logins should be allowed.
- ▶ A panel of experts on a given topic should be available for questions.
- ▶ Questions should be addressed to the board, one at a time.
- ▶ The board members should be able to attend the session anywhere, independent of physical location.
- ▶ Any board member should have the ability to remove a customer from the session, if the customer is not abiding by the rules of the session.
- ▶ Board members must have control over which questions they choose to respond to.

How would we implement this functionality?

Based on the requirements above, we will implement a solution design that we will call the “The Town Hall”.

The Town Hall consists of online sessions with customers, on scheduled dates within each month. The objective is to provide customers with an easy, cheap, and effective channel to solve their most diverse problems.

The other client is designed for the board members to use. This client allows all the features requested previously.

The functionality of the Town Hall solution is based on Places and sections features of the Sametime 3.1 Community Server Toolkit. As mentioned before, this is one of the most powerful features of the Sametime development environment, allowing the creation of virtual Places divided in sections, each with his own properties and security criteria.

To review functionality of the Sametime Places Architecture, please refer to Chapter 8, “Places and Place awareness” on page 183.

Figure 13-24 on page 415 shows the basic Places architecture of Town Hall.

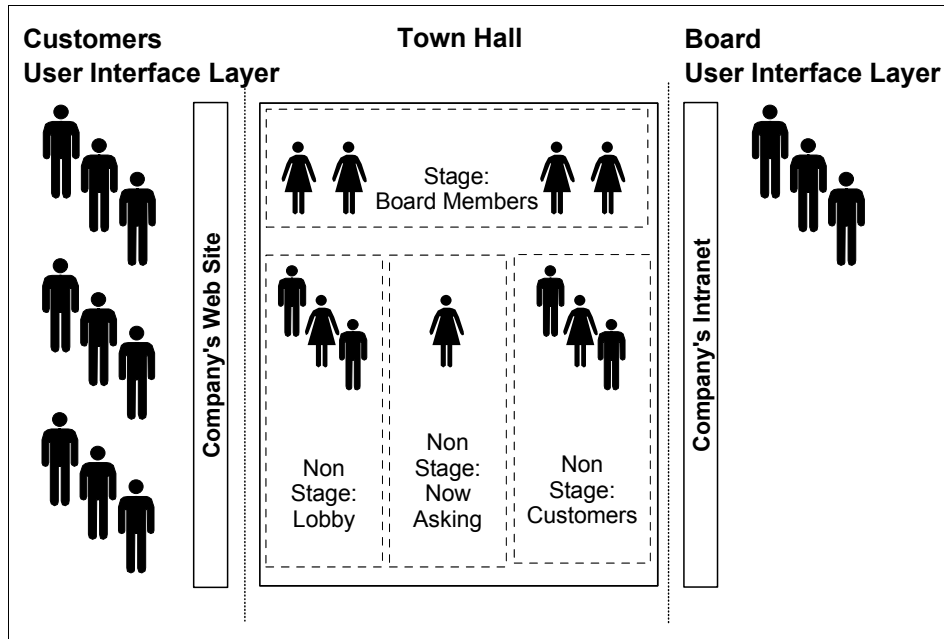


Figure 13-24 Town Hall solution's Places architecture

For people familiar with Sametime solutions, the Town Hall example is somewhat similar to the Sametime Moderated Chat offering from IBM Software Services for Lotus. The difference between the Moderated Chat and the Town Hall, however, is that the Town Hall is a “self moderated chat”. It does not require a lead moderator to decide which questions should or should not be addressed to the board.

Important: The Town Hall example discussed here is primarily based on the UN Discussion Panel example developed in Chapter 8, “Places and Place awareness” on page 183. Once the concepts and implementation method for Places and sections is understood, the focus is on adapting the context and interface to apply to the Town Hall example.

Implementing the board members client

After reviewing the requirements for the board members, we must be able to offer the following facilities to the board members' client interface:

- ▶ Ability to keep conversations between the board members while responding to customers questions on a separated area
- ▶ Ability to select the next customer on the queue to post questions

Figure 13-25 illustrates the interface structure planned for the board members client.

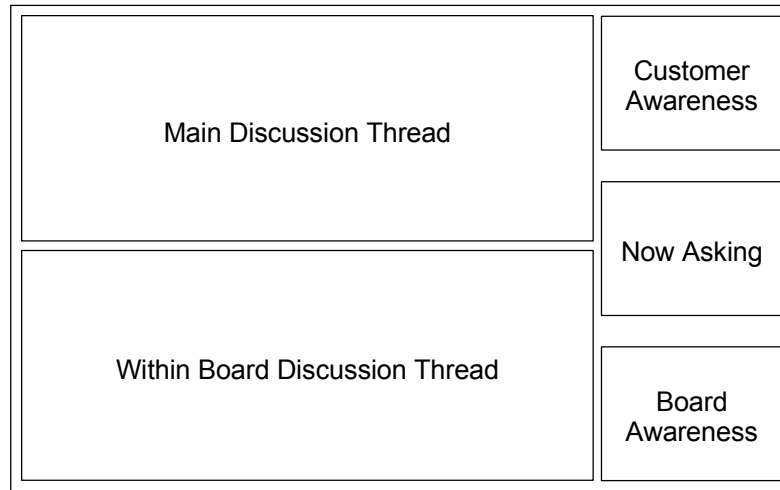


Figure 13-25 Board members client diagram

The components of the Place shown in Figure 13-25 are as follows:

- ▶ Within the main discussion thread, customers and board members are able to see everything that is going on between the board members and the customer who are in the “Now Asking” section.
- ▶ Customer awareness and board awareness represent respectively the customers list of users and the board members list names.
- ▶ The “Within Board Discussion Thread” represents the area where board members are able to keep conversations without the ability of customers to see.

Implementing the customers’ client

The customers’ client interface is simpler from the perspective that they have only one discussion thread to be aware of. The customer client interface must accomplish the following features:

- ▶ Log on or just watch the session as an anonymous user
- ▶ Raise their hands in order to show intention on posting a question
- ▶ Chat individually with the board
- ▶ See the names and profiles of board members

Figure 13-26 on page 417 illustrates the interface structure planned for the customers client.

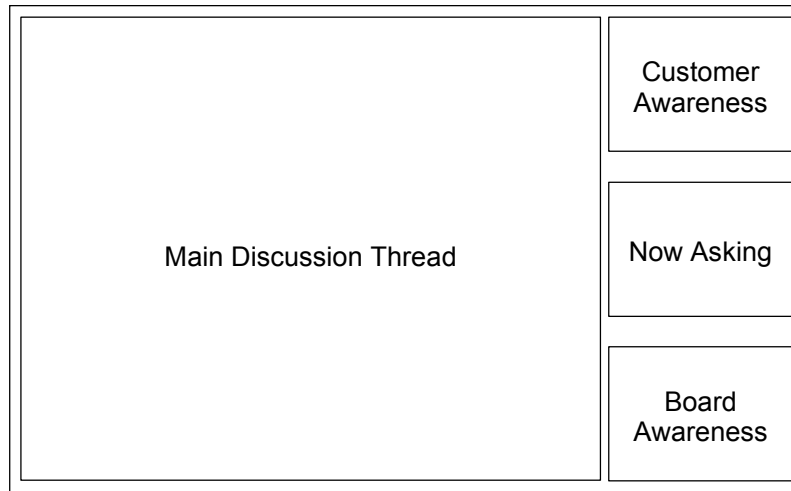


Figure 13-26 Customer client diagram

The only difference between the Client Interface diagram and the board members client interface diagram is the unique discussion thread interface.

As mentioned before, the Town Hall solution is fully based on Places and sections techniques. In order to implement this, please refer to Chapter 8, “Places and Place awareness” on page 183. There, you find comprehensive information sufficient to deploy the Town Hall solution, using the UN Discussion Panel example.

13.7.6 Implementing multilanguage educational sessions

The Town Hall solution was implemented with success and is being used frequently within FME. As a future enhancement, FME wishes to incorporate language translation capabilities to provide scheduled educational sessions with an international customer base. FME has customers around the globe, with the largest international customer base in countries such as Spain, France, and Brazil. For these customers, FME wants the Town Hall solution to implement a translation server.

The new requirement is that a customer must be able to write a question in their native language, then also receive the answers in their native language. However, the board members receive the customer questions and respond to them in English.

Implementing language translation through WebSphere Translation Server

In order to make the Town Hall solution a language abstract tool, we must plug some components into a translation server solution.

Within this specific example, we decided to use the IBM WebSphere Translation Server. Through WebSphere Translation Server, we can send text streams along with parameters such as the source and target language, and then receive the text translated to the target language.

To plug the Town Hall solution into the WebSphere Translation Server, we must implement a Place activity that talks to the WebSphere Translation Server, passes the language properties, and then gets the translated response. This activity serves as an intermediary for each chat between a customer and the board members.

In order to make this activity talk to WebSphere Translation Server, we must use the IBM WebSphere Translation Server API. More information on the WebSphere Translation Server API can be found at:

http://www-3.ibm.com/software/pervasive/ws_translation_server/

Important: This redbook does not cover any technical aspect of IBM WebSphere Translation Server. The objective here is to show the possibilities around using Sametime techniques to acquire a multilanguage solution, as described here.

Figure 13-27 on page 419 illustrates the basic architecture of the new language abstract Town Hall solution.

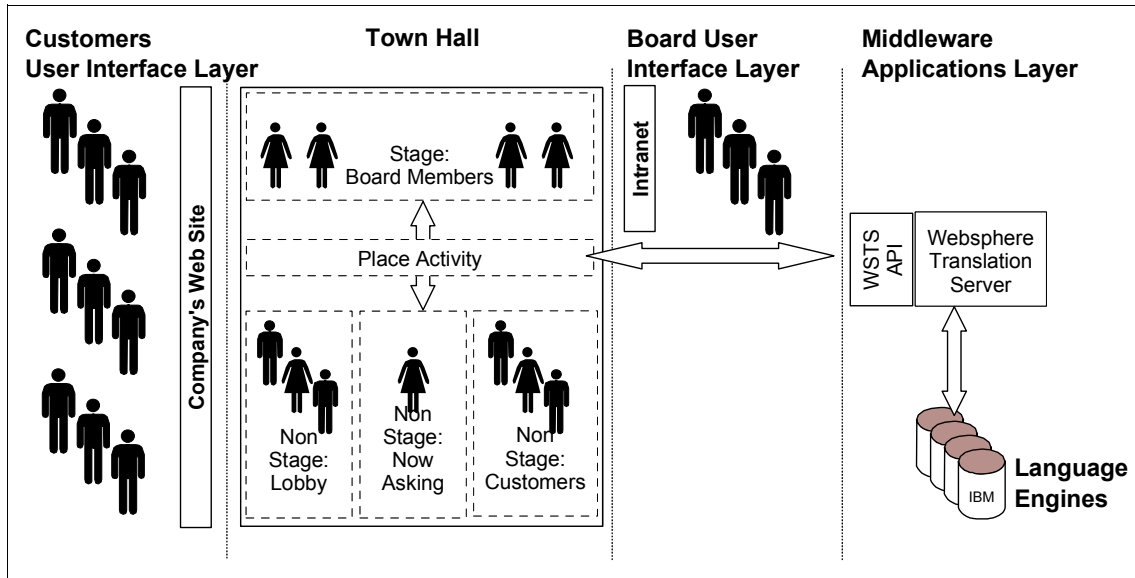


Figure 13-27 Town Hall solution's architecture with language hook

Interface modifications

To accomplish the language translation capability for the Town Hall solution, we need to modify some interface attributes. The goal is to make the user interface also language specific, based on specific language attributes.

Figure 13-28 on page 420 illustrates the interface structure.

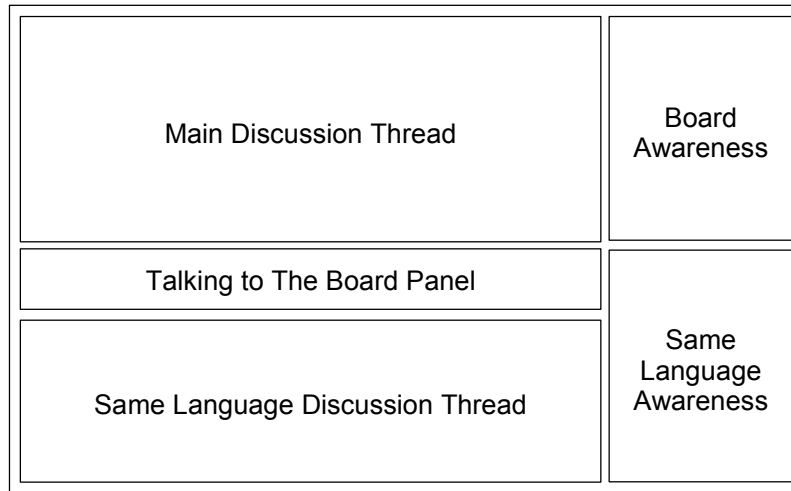


Figure 13-28 Language adapted Town Hall user client structure

- ▶ The Main Discussion Thread holds, on the language of the user, every chat kept by the board with any other customer.
- ▶ The “Talking To the Board” panel becomes available when the user is allowed to talk.
- ▶ On the “Same Language Discussion Thread”, the users of the same language can freely chat without disturbing the conversation with the board.
- ▶ The “Board Awareness” shows the names of the board members.
- ▶ On the “Same Language Awareness”, the users are able to see who is online within his same language preference.

Implementing the Place activity

Attention: General considerations regarding how to implement a Place activity can be found in 8.5, “Building the server-side” on page 203.

As mentioned before, this activity has a hook to the WebSphere Translation Server. This integration is made through the WebSphere Translation Server Java API. Example 13-13 on page 421 illustrates sample code showing how to make a call to WebSphere Translation Server in order to translate the string “The ski is blue”.

```
import com.ibm.lt.*;
public class phrase {
    public static void main(String args[]){
        LTInterface service=null;
        Object handle=null;
        String answer;
        if (args.length!=3) {
            System.out.println("Usage:");
            System.out.println(" phrase {hostname} {language} {sentence} ");
            System.out.println(" eg:\n");
            System.out.println(" phrase 127.0.0.1 enes \"The sky is blue.\" ");
            System.exit(0);
        }
        try {
            service = (LTInterface) LTEngine.GetService(args[0],args[1]);
        }catch (Throwable t){
            t.printStackTrace();
            System.out.println("no service available");
            System.exit(0);
        }
        try{
            handle=service.jltBeginTranslation("*format=text");
            answer=service.jltTranslate(handle,"The sky is blue");
            System.out.println(answer);
            service.jltEndTranslation(handle);
        }catch (Exception e){
            e.printStackTrace();
            System.exit(0);
        }
        System.exit(0);
    }
}
```

For more information on how to use the IBM WebSphere Translation Server Java API please, refer to:

http://www-3.ibm.com/software/pervasive/ws_translation_server/

Attention: When searching for translation services for Sametime, you will find references to the Lotus Translation API or Lotus Translation Server for Sametime. Please be aware that these products are no longer distributed or supported.

13.8 Phase 4: Future planned enhancements

As we have shown in this chapter, FME has implemented many of the Sametime capabilities described throughout this redbook to leverage real-time collaborative capabilities. These Sametime solutions have helped FME to become more productive and more effectively address their customers' needs.

As a final step, FME is considering how to provide a more consistent interface to allow their agents to access all of their different applications from within a consistent UI. Ultimately, FME is considering whether a portal based interface might provide a more intuitive and efficient UI for their agents.

13.8.1 Consolidating the tools interface

As mentioned earlier in the scenario, FME has multiple internal applications that are used by all divisions of the company. Depending upon which unit an agent is a member of, usage patterns and access requirements for the applications may be different.

FME is considering a Portal solution, based on IBM's WebSphere Portal Server to provide a consistent interface and easier accessibility to their disparate applications.

Considerations for a portal solution

Basically, what FME needs is a interface consolidation. Every single application is functional and productive. However, separated windows for each application affect usability.

Figure 13-29 on page 423 shows a good example of a portalized solution.



Figure 13-29 Example of a portalized solution

From within a single interface, a user has access to multiple sources of information and applications. Additionally, portlets can be Sametime-enabled, as discussed in Chapter 10, "Sametime-enabling portlets" on page 273, thereby maintaining real-time collaborative capabilities within the portal environment.

At this point, FME is in the process of defining the requirements for their portal needs. Within the call center unit, people need to have access to the following applications:

- Customer tracking application
- Knowledge base database
- Incident tracking database
- The call center attendant application

To address this needs, we decided to implement an interface similar to Figure 13-30 on page 424.

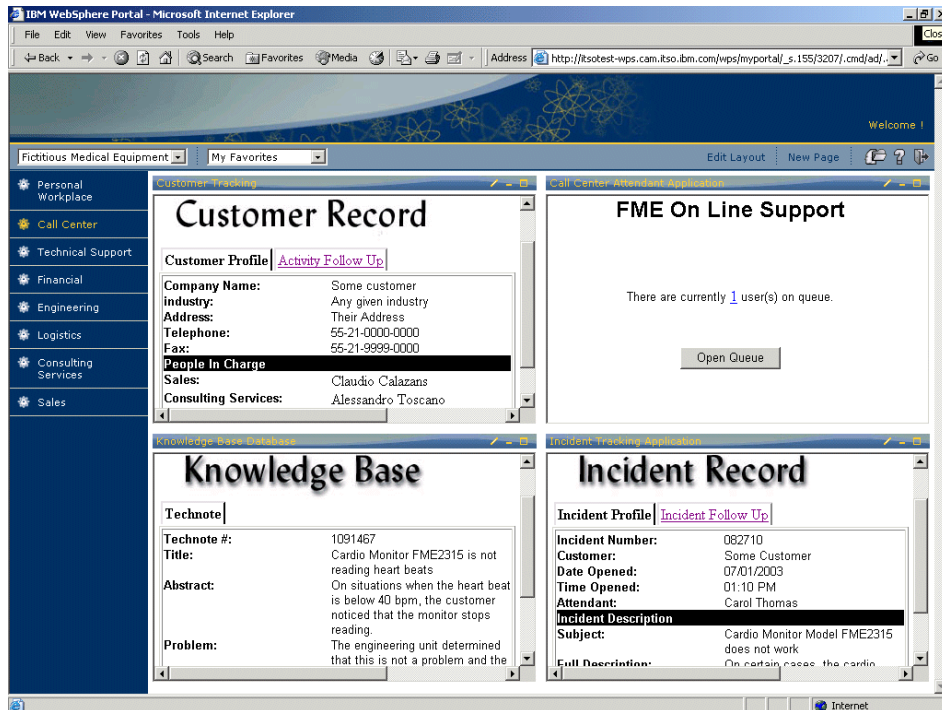


Figure 13-30 The Call Center Workplace

Attention: Since this redbook is not intended to address portlet development techniques, we do not go into specific implementation details on this subject. For more information on portlet development, please refer to the following sources:

- ▶ IBM WebSphere Portal Development Community Web site at <http://www.ibm.com/software/websphere/portal>. Once there, look for the link called WebSphere Portal zone for developers.
- ▶ *Portalizing Domino Applications for WebSphere Portal*, SG24-7004
- ▶ *How to Portalize Your Existing Web Applications*, TIPS0028, found at: <http://www.redbooks.ibm.com>
- ▶ *IBM WebSphere Portal V4 Developer's Handbook*, SG24-6897
- ▶ *IBM WebSphere Portal Version 4.1 Handbook Volumes 1, 2, and 3*, SG24-6883, SG24-6920, and SG24-6921, respectively.

13.9 Conclusion

This chapter has illustrated a business scenario that incorporates many of the Sametime features. Each section of this chapter has hopefully shown the possibilities available within Sametime to help improve collaborative capabilities, greater efficiency, and productivity within your organization. While this chapter is not intended to focus on detailed specifics of implementation, we hope that the techniques shown here can be a starting point to expand your thoughts about instant collaboration within a business context.



Part 4

Appendixes

The following two appendixes illustrate additional application examples and development approaches.



A

Visualizing Sametime

This appendix illustrates an interesting application which has been developed to illustrate a visual diagram of Sametime instant messaging traffic within a network. It has been provided by Darren M. Shaw of IBM.

While this application has been developed as a proof of concept to gain further understanding in the field of social network analysis, it provides value to application developers in several ways. First, the application illustrates a unique architectural design that integrates Sametime, IBM's Intelligent Miner™ Visualization API, Web Services, and both a custom Visualizing Sametime Client and Visualizing Sametime Server. The Sametime development community can hopefully benefit from understanding how and why the application was designed.

Perhaps more importantly, however, this application reveals a valuable set of lessons which may apply when building *any* Sametime-enabled application. The considerations should be taken into account and are found in "Lessons learned" on page 454.

Overview

Social network analysis and the ideas around it have been around for many years. A major problem in the field has been in collecting the raw data. Social networks revolve around social communication, which is innately difficult to record. Interactions between two individuals (or a group of individuals) meeting in person are difficult to record, and it is practically impossible to automate such recording. Electronic communication is, however, intrinsically easy to record. As more of the world's communication takes electronic form, an opportunity arises to record an increasing amount of human interaction.

Instant messaging is the ideal electronic communication medium over which to extract information about human interaction. Unlike e-mail, messages exchanged between people are nearly always of interest to those people. E-mails are often sent out *en masse* to many different people, countless of whom will have little or no interest in its content. IM communication is more informal, and people use it in place of face-to-face or telephone communication, which means it is likely to be used for a wider variety of communication than e-mail.

Visualizing Sametime (VS) was designed as a proof of concept system to show that social network graphs and analysis could be generated from instant messaging (IM) traffic.

Architecture

The following sections discuss the different components and overall architecture of the Visualizing Sametime system.

Outline

The Visualizing Sametime (VS) system is based on a client/server model and consists of three primary components (Figure A-1 on page 431):

1. Visualizing Sametime client
2. Visualizing Sametime server
3. Regular Sametime server

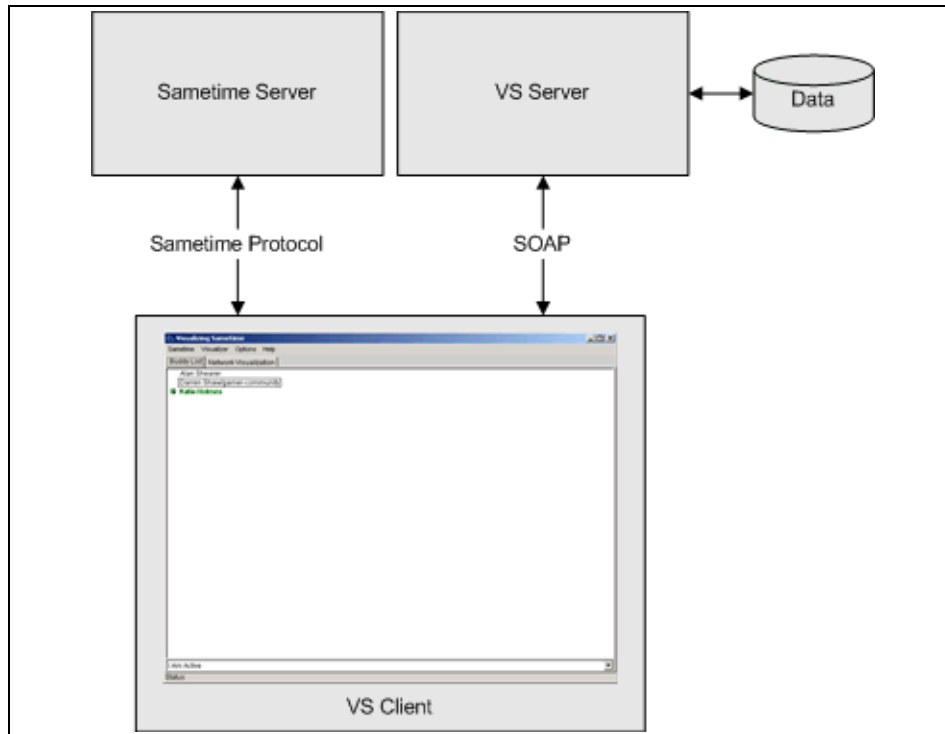


Figure A-1 Architecture of the Visualizing Sametime application

The client system provides all the basic functionality for Sametime instant messaging, buddylists and IM session handling. In addition, it also records details of all chat sessions established from and to the client. The client connects to two servers: a regular Sametime Server to provide IM facilities and a VS server to provide visualizing services.

The VS server does not itself provide any of the Sametime server functionality with regard to instant messaging. The VS server receives data from VS clients concerning IM sessions that have taken place. The VS server aggregates the session information from all hosts, using it in order to build up a global social network graph. The server, on request, will send this aggregated map to VS clients.

Although the data to build up the social networks could have been obtained via logging onto the Sametime server, the system was designed to collect data from the client. The reason for this was two-fold:

1. The server logs may not be created, stored, or be accessible.

2. Logging data on a specially adapted client provides an easy way for a user to opt in or out of the scheme.

Personal privacy was a concern throughout this project, so a client logging system was selected.

Visualizing Sametime client

The VS client is built in Java, using the Sametime client API to provide Sametime IM communication and the IBM Intelligent Miner Visualization 8.1 API to provide association-graphing facilities.

The VS client communicates with a Sametime server using the standard Sametime protocols. The application provides the base Sametime IM functionality used in Sametime Connect. A buddylist is provided for storing contacts and IM sessions can be joined or established from the client. All IM related traffic travels between the client and Sametime server as normal.

The VS client needs to record details of all IM sessions it is party to. This is done by adding listeners that are invoked when an IM session is established or closed. When a listener is fired, it connects to the Web service on the VS server and records the details of the IM session that has just been created or destroyed.

The VS client also uses the IBM Intelligent Miner Visualization 8.1 API. Intelligent Miner Visualization is used in the data-mining arena to graphically render data mining models. In VS, association models are used to describe the Sametime social network. Association models show items and associations between those items. In the graphical form, an association graph is shown as a series of nodes and connecting arrows (see Figure A-2 on page 433).

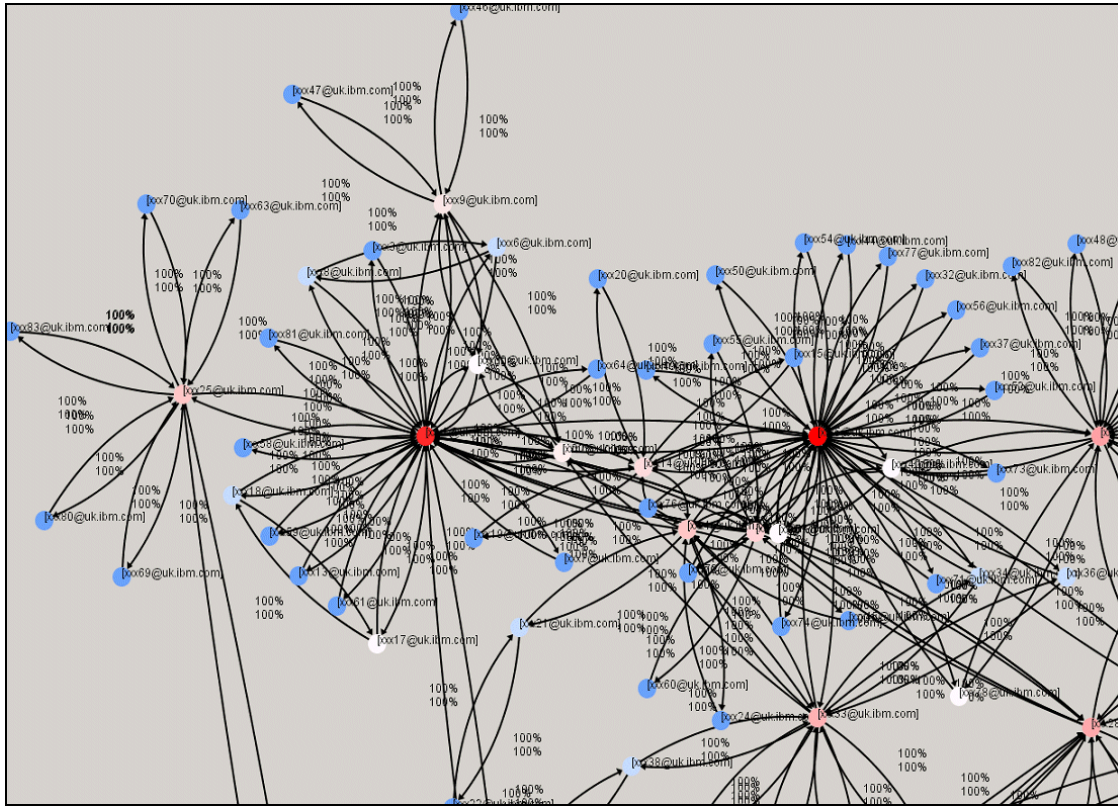


Figure A-2 Association Model of a Sametime social network

The Intelligent Miner Visualization API allows association graphs to be rendered on the VS client. Once such graphs are displayed, they can be manipulated by the user. Nodes that represent Sametime users can be hidden, filtered, or moved around. It is also possible to fan out from a node or set of nodes in order to show all nodes that the set of nodes has a direct (that is, one step/degree) connection to.

The user can refresh the social network graph; on doing so, a Web service request is sent to the VS server requesting that the latest PMML document be returned. On receiving the PMML, the client uses the classed provided by the Intelligent Miner Visualization API to render a graph and display it.

Implementation

The application was developed in WebSphere Studio Application Developer 5.0. The VS server component development was greatly assisted by WebSphere Studio Application Developer's application server test environment.

GUI Design

The GUI for the VS client consists of a window with a two page tabbed panel. The first page holds a simple buddylist panel (Figure A-3).

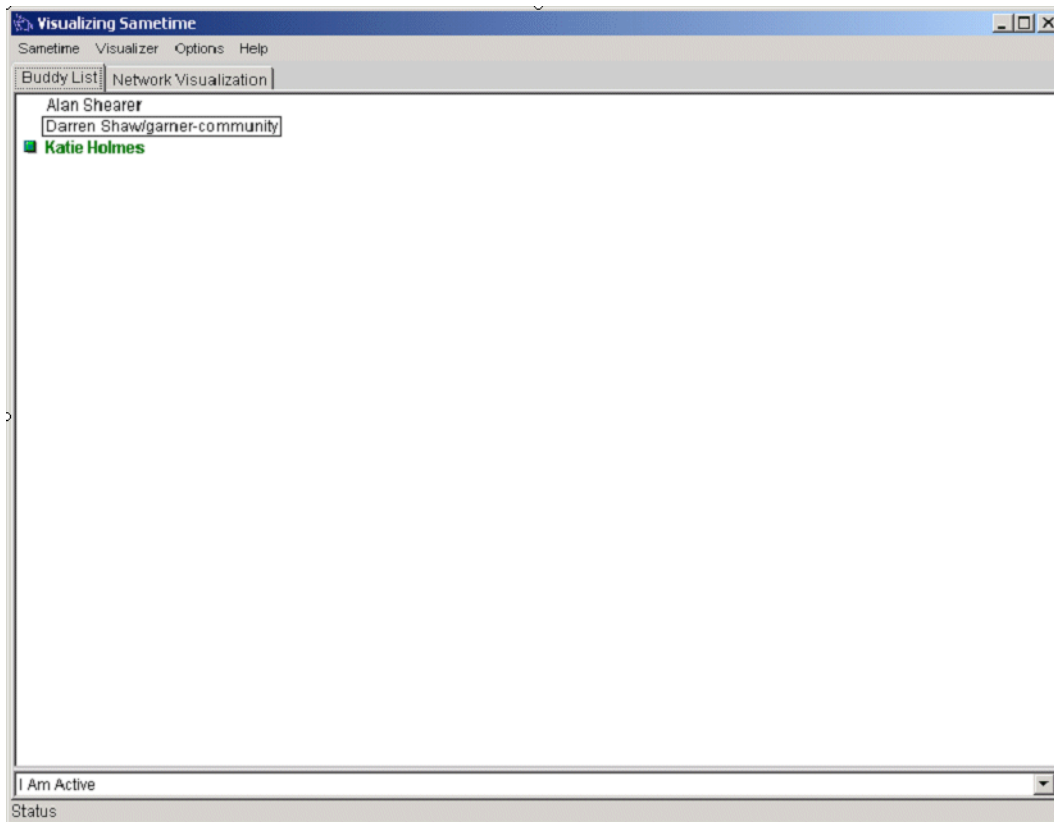


Figure A-3 Visualizing Sametime GUI

The second tab acts as the network visualization page. From within this tab, the user has the choice to see a graphic image of the network connections, as shown in Figure A-4 on page 435.

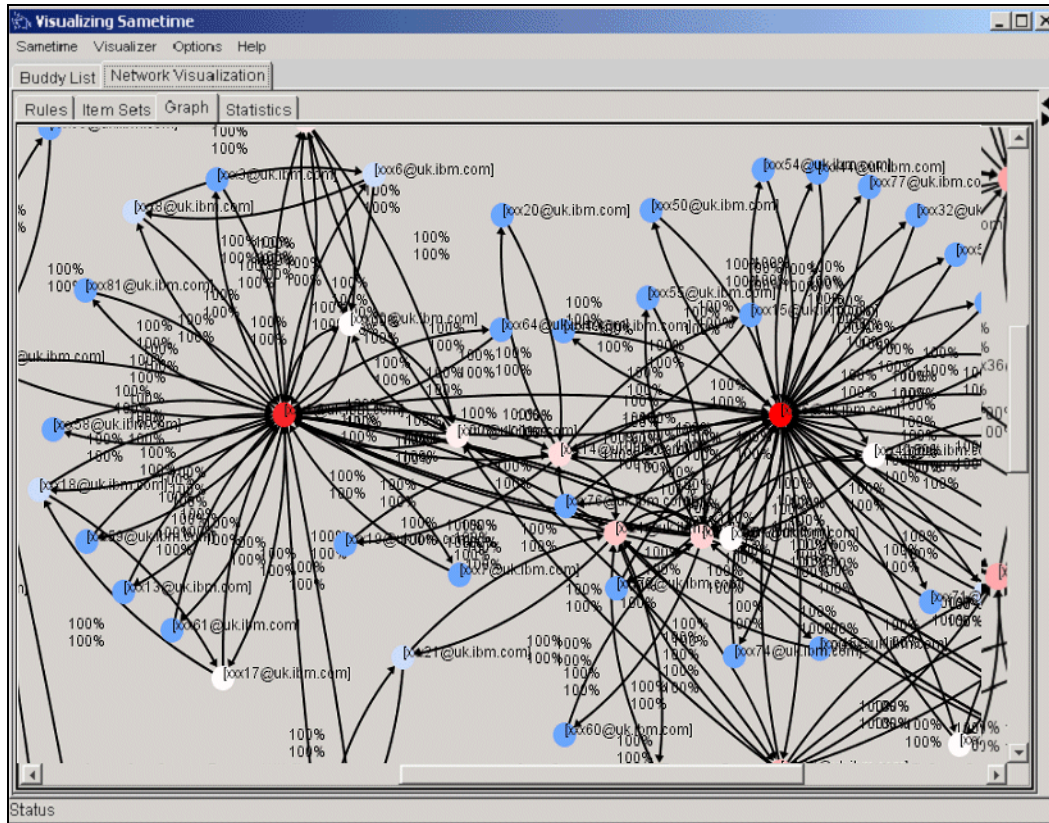


Figure A-4 Network Visualization Tab

Web services

A Web service proxy class called `ServicesProxy` was generated by WebSphere Studio Application Developer for the VS client to use to communicate with the VS server (see Example A-1). The class includes three key methods: `logOpenSession`, which logs the opening of an IM session, `logCloseSession`, which logs the closing of an IM session, and `generatePMML`, which requests the PMML document describing the network.

Example: A-1 ServicesProxy.java (VS Client)

```
package com.ibm.hursley.sametimeclient.proxy;

import java.net.*;
import java.util.*;
import org.w3c.dom.*;
import org.apache.soap.*;
```

```

import org.apache.soap.encoding.*;
import org.apache.soap.encoding.soapenc.*;
import org.apache.soap.rpc.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.messaging.*;

public class ServicesProxy
{
    private Call call = createCall();
    private URL url = null;
    private String stringURL =
"http://localhost:8080/SametimeDataServerWA/servlet/rpcrouter";

    public ServicesProxy(String url){
        this.stringURL = url + "/SametimeDataServerWA/servlet/rpcrouter";
    }

    public synchronized void setEndPoint(URL url)
    {
        this.url = url;
    }

    public synchronized URL getEndPoint() throws MalformedURLException
    {
        return getURL();
    }

    private URL getURL() throws MalformedURLException
    {
        {
            if (url == null && stringURL != null && stringURL.length() > 0)
            {
                url = new URL(stringURL);
            }
            return url;
        }
    }

    public synchronized void logCloseSession(java.lang.String
sourceUser,java.lang.String destinationUser) throws Exception
    {
        String targetObjectURI =
"http://tempuri.org/com.ibm.hursley.sametimedataserver.services.Services";
        String SOAPActionURI = "";

        if(getURL() == null)
        {
            throw new SOAPException(Constants.FAULT_CODE_CLIENT,
"A URL must be specified via ServicesProxy.setEndPoint(URL).");
        }
    }

```

```

        call.setMethodName("logCloseSession");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        call.setTargetObjectURI(targetObjectURI);
        Vector params = new Vector();
        Parameter sourceUserParam = new Parameter("sourceUser",
java.lang.String.class, sourceUser, Constants.NS_URI_SOAP_ENC);
        params.addElement(sourceUserParam);
        Parameter destinationUserParam = new Parameter("destinationUser",
java.lang.String.class, destinationUser, Constants.NS_URI_SOAP_ENC);
        params.addElement(destinationUserParam);
        call.setParams(params);
        Response resp = call.invoke(getURL(), SOAPActionURI);

        //Check the response.
        if (resp.generatedFault())
        {
            Fault fault = resp.getFault();
            call.setFullTargetObjectURI(targetObjectURI);
            throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
        }
    }

    public synchronized void logOpenSession(java.lang.String
sourceUser,java.lang.String destinationUser) throws Exception
    {
        String targetObjectURI =
"http://tempuri.org/com.ibm.hursley.sametimedataserver.services.Services";
        String SOAPActionURI = "";

        if(getURL() == null)
        {
            throw new SOAPException(Constants.FAULT_CODE_CLIENT,
            "A URL must be specified via ServicesProxy.setEndPoint(URL).");
        }

        call.setMethodName("logOpenSession");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        call.setTargetObjectURI(targetObjectURI);
        Vector params = new Vector();
        Parameter sourceUserParam = new Parameter("sourceUser",
java.lang.String.class, sourceUser, Constants.NS_URI_SOAP_ENC);
        params.addElement(sourceUserParam);
        Parameter destinationUserParam = new Parameter("destinationUser",
java.lang.String.class, destinationUser, Constants.NS_URI_SOAP_ENC);
        params.addElement(destinationUserParam);
        call.setParams(params);
        Response resp = call.invoke(getURL(), SOAPActionURI);

```

```

        //Check the response.
        if (resp.generatedFault())
        {
            Fault fault = resp.getFault();
            call.setFullTargetObjectURI(targetObjectURI);
            throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
        }
    }

    public synchronized java.lang.String generatePMML(int minutes) throws
    Exception
    {
        String targetObjectURI =
        "http://tempuri.org/com.ibm.hursley.sametimedataserver.services.Services";
        String SOAPActionURI = "";

        if(getURL() == null)
        {
            throw new SOAPException(Constants.FAULT_CODE_CLIENT,
            "A URL must be specified via ServicesProxy.setEndPoint(URL).");
        }

        call.setMethodName("generatePMML");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        call.setTargetObjectURI(targetObjectURI);
        Vector params = new Vector();
        Parameter minutesParam = new Parameter("minutes", int.class, new
        Integer(minutes), Constants.NS_URI_SOAP_ENC);
        params.addElement(minutesParam);
        call.setParams(params);
        Response resp = call.invoke(getURL(), SOAPActionURI);

        //Check the response.
        if (resp.generatedFault())
        {
            Fault fault = resp.getFault();
            call.setFullTargetObjectURI(targetObjectURI);
            throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
        }
        else
        {
            Parameter refValue = resp.getReturnValue();
            return ((java.lang.String)refValue.getValue());
        }
    }
}

```



```

private static Call createCall()
{
    Call call = new Call();
    SOAPMappingRegistry smr = call.getSOAPMappingRegistry();
    return call;
}
}

```

The `logOpenSession` and `logCloseSession` methods in the `ServicesProxy` class are called from listener objects run when an IM session is created or destroyed on the VS client. The `ServiceProxy` object is created and its methods executed from a separate thread so that other operations of the VS client are not locked while client-server communication takes place.

IM session data aggregation

Aggregation of IM session data takes place on the VS server. Data provided through Web service calls to indicate the creation or closing of IM sessions is sent to the `addSessionData` method in the `DatabaseStoreKeeper` class (Example A-2), which is responsible for writing the IM session data into the back-end database.

Example: A-2 DatabaseStoreKeeper.java (VS Server)

```

public class DatabaseStoreKeeper {

    private static DatabaseStoreKeeper instance = null;

    public DatabaseStoreKeeper() {
        super();
    }

    public static DatabaseStoreKeeper getInstance() {

        if (instance == null) {
            instance = new DatabaseStoreKeeper();
        }

        return instance;
    }

    public void addSessionData(String sourceUser, String destinationUser,
String operation) {

        String query = "";
        Connection connection = null;

```

```

        PreparedStatement statement = null;

        try {
            // get connection to database
            connection = Database.getInstance().getConnection();

            query = "INSERT INTO " + Database.SAMETIME_SESSIONS_TABLE + "(DATE,
SOURCE_USER, DESTINATION_USER, OPERATION) VALUES (?, ?, ?, ?)";

            java.util.Date now = new java.util.Date();

            statement = connection.prepareStatement(query);
            statement.setTimestamp(1, new Timestamp(now.getTime()));
            statement.setString(2, sourceUser);
            statement.setString(3, destinationUser);
            statement.setString(4, operation);

            statement.executeUpdate();
            statement.close();
            connection.close();
        }
        catch (Exception e) {
            try {
                statement.close();
                connection.close();
            }
            catch (Exception e2){

            }

            System.out.println("Exception adding session data to
sametime.sessions table: " + e.toString());
        }
    }

    public Vector getSessions(java.util.Date startDate, java.util.Date endDate)
    {
        Vector sessions = new Vector();

        String query = "";
        Connection connection = null;
        PreparedStatement statement = null;
        ResultSet results = null;

        try {
            connection = Database.getInstance().getConnection();
            query = "SELECT * FROM " + Database.SAMETIME_SESSIONS_TABLE ;

            statement = connection.prepareStatement(query);

```

```

        results = statement.executeQuery();

        while(results.next()) {
            java.util.Date resultDate = results.getDate(2);

            String sourceUser = cleanUserID(results.getString(3));
            String destinationUser = cleanUserID(results.getString(4));
            String operation = results.getString(5);
            Session session = new Session(resultDate, sourceUser,
destinationUser, operation);
            sessions.add(session);

        }
        results.close();
        statement.close();
        connection.close();
    }
    catch(Exception e) {
        System.out.println("Exception in DatabaseStoreKeeper.getSessions: " +
e.toString());
        try {
            results.close();
            statement.close();
            connection.close();
        }
        catch(Exception e2) {

        }
    }
}

return sessions;
}

private String cleanUserID(String userID) {
    String cleanedString = "";

    if(userID.indexOf(" ") > -1) {
        userID = userID.substring(0,userID.indexOf(" "));
    }

    userID = userID.toLowerCase();

    return userID;
}

}

```

The DatabaseStoreKeeper class also contains a method called getSession, which is used to extract the IM session data from the database into a vector of Session objects, used in the aggregation of IM session data. The getSession method has parameters for start and end dates, which will eventually allow the sessions to be selected with a restricted date range. This functionality has yet to be implemented.

The getSession method is called from a Miner object, which builds up a model from the IM session data and produces a PMML document (see Example A-3) to represent the model.

The PMML document is made up of four main sections. AssocInputStats describe the statistics that give an outline of the model. AssocItems define the items that are held within the model (in this case, individual Sametime users). AssocItemSets define the nodes that are displayed on the graph and are composed of a set of AssocItems; in this case, however an AssocItem only ever has a single AssocItem. Finally, AssocRule defines an association between two AssocItemSets, in this case the link between two Sametime users. An example of this PMML Model is shown in Example A-3.

Example: A-3 Sample PMML model

```
<?xml version="1.0" encoding="iso-8859-1" ?><?idm IDM-RESULT-TYPE="10" ?><PMML
version="1.1"><Header copyright="Copyright IBM Corp. 2000, All Rights Reserved"/>
<DataDictionary numberOfFields="1"><DataField name="item"
optype="categorical"/></DataDictionary>
<AssociationModel><AssocInputStats numberOfTransactions="0" numberOfItems="85"
minimumSupport="0.000000" minimumConfidence="0.000000" numberOfItemsets="85"
numberOfRules="222"/>
<AssocItem id="0" value="xxx1@uk.ibm.com"/>
<AssocItem id="1" value="xxx2@uk.ibm.com"/>
<AssocItem id="2" value="xxx3@uk.ibm.com"/>
<AssocItem id="3" value="xxx4@uk.ibm.com"/>
<AssocItem id="4" value="xxx5@uk.ibm.com"/>
<AssocItem id="5" value="xxx6@uk.ibm.com"/>
<AssocItem id="6" value="xxx7@uk.ibm.com"/>
<AssocItem id="7" value="xxx8@uk.ibm.com"/>
<AssocItem id="8" value="xxx9@uk.ibm.com"/>
<AssocItem id="9" value="xxx10@uk.ibm.com"/>
<AssocItem id="10" value="xxx11@uk.ibm.com"/>
<AssocItem id="11" value="xxx12@uk.ibm.com"/>
<AssocItem id="12" value="xxx13@uk.ibm.com"/>
<AssocItem id="13" value="xxx14@uk.ibm.com"/>
<AssocItem id="14" value="xxx15@uk.ibm.com"/>
<AssocItem id="15" value="xxx16@uk.ibm.com"/>
<AssocItem id="16" value="xxx17@uk.ibm.com"/>
<AssocItem id="17" value="xxx18@uk.ibm.com"/>
<AssocItem id="18" value="xxx19@uk.ibm.com"/>
```

```
<AssocItem id="19" value="xxx20@uk.ibm.com"/>
<AssocItem id="20" value="xxx21@uk.ibm.com"/>
<AssocItem id="21" value="xxx22@uk.ibm.com"/>
<AssocItem id="22" value="xxx23@uk.ibm.com"/>
<AssocItem id="23" value="xxx24@uk.ibm.com"/>
<AssocItem id="24" value="xxx25@uk.ibm.com"/>
<AssocItem id="25" value="xxx26@uk.ibm.com"/>
<AssocItem id="26" value="xxx27@uk.ibm.com"/>
<AssocItem id="27" value="xxx28@uk.ibm.com"/>
<AssocItem id="28" value="xxx29@uk.ibm.com"/>
<AssocItem id="29" value="xxx30@uk.ibm.com"/>
<AssocItem id="30" value="xxx31@uk.ibm.com"/>
<AssocItem id="31" value="xxx32@uk.ibm.com"/>
<AssocItem id="32" value="xxx33@uk.ibm.com"/>
<AssocItem id="33" value="xxx34@uk.ibm.com"/>
<AssocItem id="34" value="xxx35@uk.ibm.com"/>
<AssocItem id="35" value="xxx36@uk.ibm.com"/>
<AssocItem id="36" value="xxx37@uk.ibm.com"/>
<AssocItem id="37" value="xxx38@uk.ibm.com"/>
<AssocItem id="38" value="xxx39@uk.ibm.com"/>
<AssocItem id="39" value="xxx40@uk.ibm.com"/>
<AssocItem id="40" value="xxx41@uk.ibm.com"/>
<AssocItem id="41" value="xxx42@uk.ibm.com"/>
<AssocItem id="42" value="xxx43@uk.ibm.com"/>
<AssocItem id="43" value="xxx44@uk.ibm.com"/>
<AssocItem id="44" value="xxx45@uk.ibm.com"/>
<AssocItem id="45" value="xxx46@uk.ibm.com"/>
<AssocItem id="46" value="xxx47@uk.ibm.com"/>
<AssocItem id="47" value="xxx48@uk.ibm.com"/>
<AssocItem id="48" value="xxx49@uk.ibm.com"/>
<AssocItem id="49" value="xxx50@uk.ibm.com"/>
<AssocItem id="50" value="xxx51@uk.ibm.com"/>
<AssocItem id="51" value="xxx52@uk.ibm.com"/>
<AssocItem id="52" value="xxx53@uk.ibm.com"/>
<AssocItem id="53" value="xxx54@uk.ibm.com"/>
<AssocItem id="54" value="xxx55@uk.ibm.com"/>
<AssocItem id="55" value="xxx56@uk.ibm.com"/>
<AssocItem id="56" value="xxx57@uk.ibm.com"/>
<AssocItem id="57" value="xxx58@uk.ibm.com"/>
<AssocItem id="58" value="xxx59@uk.ibm.com"/>
<AssocItem id="59" value="xxx60@uk.ibm.com"/>
<AssocItem id="60" value="xxx61@uk.ibm.com"/>
<AssocItem id="61" value="xxx62@uk.ibm.com"/>
<AssocItem id="62" value="xxx63@uk.ibm.com"/>
<AssocItem id="63" value="xxx64@uk.ibm.com"/>
<AssocItem id="64" value="xxx65@uk.ibm.com"/>
<AssocItem id="65" value="xxx66@uk.ibm.com"/>
<AssocItem id="66" value="xxx67@uk.ibm.com"/>
<AssocItem id="67" value="xxx68@uk.ibm.com"/>
```

```

<AssocItem id="68" value="xxx69@uk.ibm.com"/>
<AssocItem id="69" value="xxx70@uk.ibm.com"/>
<AssocItem id="70" value="xxx71@uk.ibm.com"/>
<AssocItem id="71" value="xxx72@uk.ibm.com"/>
<AssocItem id="72" value="xxx73@uk.ibm.com"/>
<AssocItem id="73" value="xxx74@uk.ibm.com"/>
<AssocItem id="74" value="xxx75@uk.ibm.com"/>
<AssocItem id="75" value="xxx76@uk.ibm.com"/>
<AssocItem id="76" value="xxx77@uk.ibm.com"/>
<AssocItem id="77" value="xxx78@uk.ibm.com"/>
<AssocItem id="78" value="xxx79@uk.ibm.com"/>
<AssocItem id="79" value="xxx80@uk.ibm.com"/>
<AssocItem id="80" value="xxx81@uk.ibm.com"/>
<AssocItem id="81" value="xxx82@uk.ibm.com"/>
<AssocItem id="82" value="xxx83@uk.ibm.com"/>
<AssocItem id="83" value="xxx84@uk.ibm.com"/>
<AssocItem id="84" value="xxx85@uk.ibm.com"/>
<AssocItemset id="0" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="0"
/></AssocItemset>
<AssocItemset id="1" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="1"
/></AssocItemset>
<AssocItemset id="2" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="2"
/></AssocItemset>
<AssocItemset id="3" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="3"
/></AssocItemset>
<AssocItemset id="4" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="4"
/></AssocItemset>
<AssocItemset id="5" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="5"
/></AssocItemset>
<AssocItemset id="6" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="6"
/></AssocItemset>
<AssocItemset id="7" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="7"
/></AssocItemset>
<AssocItemset id="8" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="8"
/></AssocItemset>
<AssocItemset id="9" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="9"
/></AssocItemset>
<AssocItemset id="10" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="10"
/></AssocItemset>
<AssocItemset id="11" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="11"
/></AssocItemset>
<AssocItemset id="12" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="12"
/></AssocItemset>
<AssocItemset id="13" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="13"
/></AssocItemset>
<AssocItemset id="14" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="14"
/></AssocItemset>
<AssocItemset id="15" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="15"
/></AssocItemset>

```

[illegible]

```

<AssocItemset id="40" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="40"
/></AssocItemset>
<AssocItemset id="41" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="41"
/></AssocItemset>
<AssocItemset id="42" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="42"
/></AssocItemset>
<AssocItemset id="43" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="43"
/></AssocItemset>
<AssocItemset id="44" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="44"
/></AssocItemset>
<AssocItemset id="45" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="45"
/></AssocItemset>
<AssocItemset id="46" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="46"
/></AssocItemset>
<AssocItemset id="47" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="47"
/></AssocItemset>
<AssocItemset id="48" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="48"
/></AssocItemset>
<AssocItemset id="49" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="49"
/></AssocItemset>
<AssocItemset id="50" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="50"
/></AssocItemset>
<AssocItemset id="51" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="51"
/></AssocItemset>
<AssocItemset id="52" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="52"
/></AssocItemset>
<AssocItemset id="53" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="53"
/></AssocItemset>
<AssocItemset id="54" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="54"
/></AssocItemset>
<AssocItemset id="55" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="55"
/></AssocItemset>
<AssocItemset id="56" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="56"
/></AssocItemset>
<AssocItemset id="57" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="57"
/></AssocItemset>
<AssocItemset id="58" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="58"
/></AssocItemset>
<AssocItemset id="59" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="59"
/></AssocItemset>
<AssocItemset id="60" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="60"
/></AssocItemset>
<AssocItemset id="61" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="61"
/></AssocItemset>
<AssocItemset id="62" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="62"
/></AssocItemset>
<AssocItemset id="63" support="1.000000" numberOfItems="1"><AssocItemRef itemRef="63"
/></AssocItemset>

```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

<AssocRule support="1.000000" confidence="1.000000" antecedent="77" consequent="77" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="9" consequent="77" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="77" consequent="9" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="77" consequent="4" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="4" consequent="77" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="9" consequent="78" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="78" consequent="9" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="24" consequent="79" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="79" consequent="24" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="0" consequent="80" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="80" consequent="0" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="9" consequent="81" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="81" consequent="9" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="24" consequent="82" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="82" consequent="24" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="9" consequent="83" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="83" consequent="9" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="27" consequent="84" />
<AssocRule support="1.000000" confidence="1.000000" antecedent="84" consequent="27" />
</AssociationModel>
</PMML>

```

Outcome

The following sections address the outcome of the trial program and discuss the lessons learned.

Trial program

In order to test the application, it was initially distributed to just four colleagues. The trial was open to anyone, however, and it did not take long before 20-30 people were using it as their Sametime client. People saw the graphs being displayed on colleague's workstations and were curious as to what the graphs showed. A deliberate ploy was to only make the network graphs visible to those using the applications and contributing to the graph themselves. This encouraged people to use the application.

As an IM session is logged on the VS server, if either of the two participants is using the VS client, many more people appeared on the graph than were participating in the trial. Again, this was a deliberate decision in order to get as large a data set as possible. To date, around 200 people are shown on the graph, with around 500 links between them. The majority of the users are based in the UK, which is to be expected, as this is where the application originates

from, though there are a handful of users on the graph from the US, France, Germany, and New Zealand.

Results

The application caused a substantial amount of interest. On seeing the graphs, people were drawn to find out if they were on it and what did it mean if they were. The graphs proved to be of interest to look at purely from an aesthetic angle alone.

Without applying any formal social network analysis, the graphs showed a number of interesting properties:

1. Unknown common acquaintances, that is, if person A knows B and C, but had not realized B and C also knew each other directly.
2. Community hubs, that is, those who are part of several groups who act as bridges between the groups.
3. Close positioning in the graph implies common links, thus identifying people who are similarly connected.
4. Highlights routes of influence between people, showing how someone may indirectly influence a third party.

The results cannot be conclusive, as the data set was not complete or representative. Only a minute proportion of Sametime users used the system. If a larger number of users were to be used, more advanced and quantitative assessment of the graphs would have to be taken in order to extract any useful information.

The system was only designed to be a proof of concept demonstrator. It shows that social network graphs can (relatively easily) be built up from IM traffic. To make the system useful, so that it can identify communities and an individual's position within the network, more work is needed on the analysis of graphs. Work is also needed to understand the context that a link is formed in. The underlying data collection and rendering system, however, is sound.

Concerns

Informal feedback regarding VS has raised several concerns, all of them, understandably, revolving around the privacy aspect.

The system did include several restrictions in order to protect individual user's privacy. No data analysis was performed on the content of any IM session. The only thing recorded was the fact that an IM session had taken place between users. Another privacy restriction was that the amount of IM traffic between two

users on the system was not shown, either in terms of the number of IM sessions or the amount of traffic within those sessions. It would have been possible to show on the graph (perhaps by the thickness of the lines) the number of sessions between two users, but to encourage people to use the system, this data was not shown.

Any future development of VS will have to seriously consider privacy implications so that people will be willing to use the system.

Future improvements

- ▶ Adapting the VS server to use the Intelligent Miner Modelling 8.1 for data aggregation (rather than custom aggregation algorithm)
- ▶ Securing functionality to prevent data spoofing
- ▶ Setting up a temporal element to show IM traffic between specific times
- ▶ Automatic social network analysis
- ▶ Increased privacy protection

Lessons learned

Although VS is intended as a proof of concept system, a number of lessons were learned regarding the building of Sametime-enabled applications.

1. Users do not expect to lose any functionality that exists in the Sametime-based client they are currently using. Even in situations where the additional features far outweighed any loss, users still demanded all the functionality of their existing system.
2. Applications using Sametime functionality are likely to be left running on the user's desktop permanently and, therefore, need to be as lightweight as possible. The resources used by the VS client has posed a problem to some users. The overhead was mainly caused by the association graphing facilities.
3. Privacy is a big issue for people using custom Sametime applications and needs to be thought of while designing any Sametime-based application.
4. Careful use of threading needs to be used in applications built with Sametime components in. It is important that the surrounding application does not hold up the Sametime elements, as delays can be annoying to users.



Online customer support application example

The following appendix material has been contributed by Vince Fertig of IBM Software Services for Lotus (ISSL). It provides a high level overview of an online customer support application recently developed by a team internally within IBM. It provides an alternate means of handling customer support through live chats between a customer and customer support agents. The goal of this appendix is to illustrate an overview of the architecture while also highlighting some of the important design decisions and their potential benefits to the application.

Overview

In this appendix, we will discuss a larger and more complete Sametime application. This application provides an alternate means of handling customer support, through live chats between a customer and customer support agents.

A customer logs onto the system with a custom Sametime application and makes a request for assistance by selecting from one of their open problem reports. The agents use a custom Sametime application to monitor customers requesting assistance. Each agent is able to see the relevant information about a problem report (problem area, customer, and company), as well as how long the customer has been waiting for a response. At the click of a button, the agent can initiate a dialog with the customer. The customer disappears from the other agents' monitoring views and the one-on-one chat begins.

Figure B-1 illustrates both the customer and agent application communicating with both the Sametime server and a customer support database. The details of customer support database, from which the applets retrieve problem report information, are not relevant to this example.

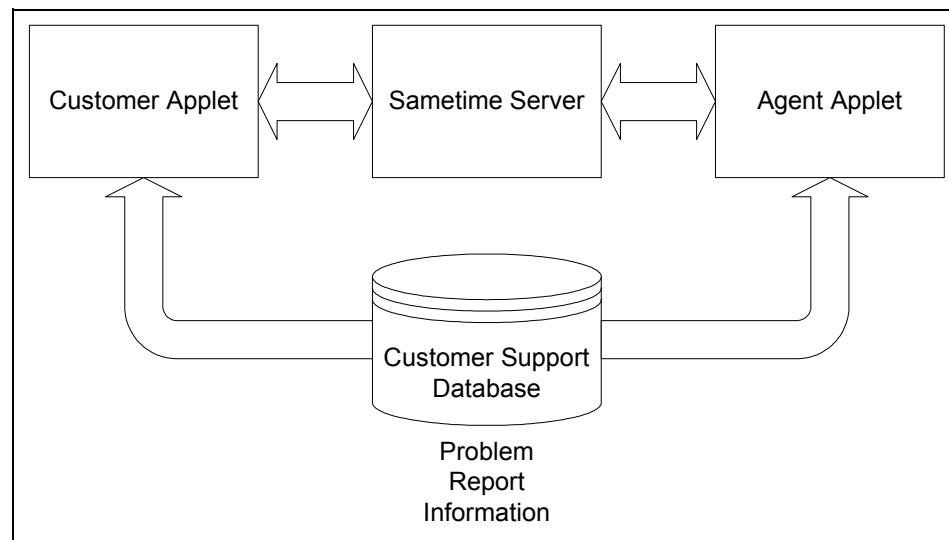


Figure B-1 Component diagram

Sametime functionality supporting this application

This application primarily makes use of the Sametime Places service. The diagram shown in Figure B-2 illustrates the structure of the support application, based on a Place made up of multiple sections.

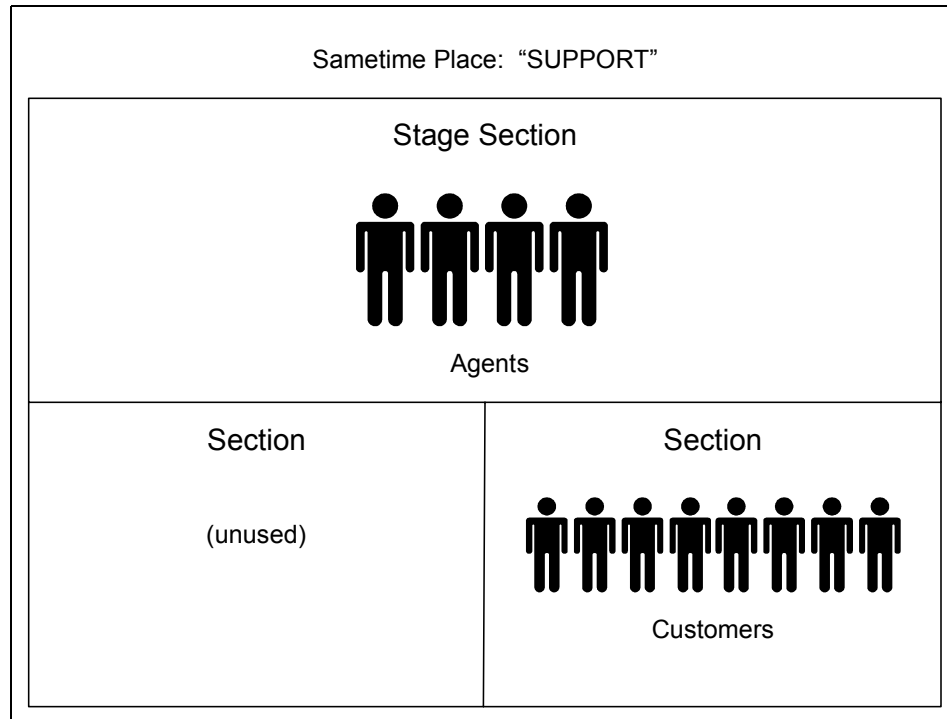


Figure B-2 Diagram of support place

The application has the following characteristics:

- ▶ All users enter a common Place.
- ▶ Agents reside in the stage section; customers reside in one of the non-stage sections.
- ▶ Communication (chat) between customers and agents takes place using the `sendText()` functionality of the `UserInPlace` objects.
- ▶ Control information passed between customers and agents (see handshake section below) takes place using the `sendData()` functionality.
- ▶ Place member attributes (`STExtendedAttributes`) specify customer and agent characteristics (customer name, company, problem area, short description, and so forth).

- ▶ Agents listen for users entering/leaving the agent and customer sections (SectionListener).
- ▶ Agents listen for attribute changes on customers and other agents (UserInPlaceListener).
- ▶ Customers do not listen for attribute changes.

We do not make use of any of the special characteristics of the stage section; however, placing customers and agents in separate sections allows us to easily identify the type of user. The communications channel between UserInPlace objects comes “for free” as part of the Places service, so we do not need to set up and use the instant messaging service.

The choice of Place member attributes for storing user characteristics is important. The Sametime server handles sending attribute information to all interested parties (UserInPlace listeners). This service saves you from having to implement some sort of protocol between the customer and agent applets for exchange of data. With server management of data, any agent that comes into the system receives all the information existing agents have. One point to note is that excessive distribution of attribute values can cause performance degradation, so one should limit listeners to minimum necessary. As noted above, agents listen for changes in customer attributes while customers do not listen to agent attributes.

The customer applet

The following section describes the customer applet, including the customer UI, the customer attributes, and the different states that may be associated with them.

Customer UI

When the customer logs onto the Web site, they are presented with a screen showing their open problem reports. This is shown in Figure B-3 on page 459.

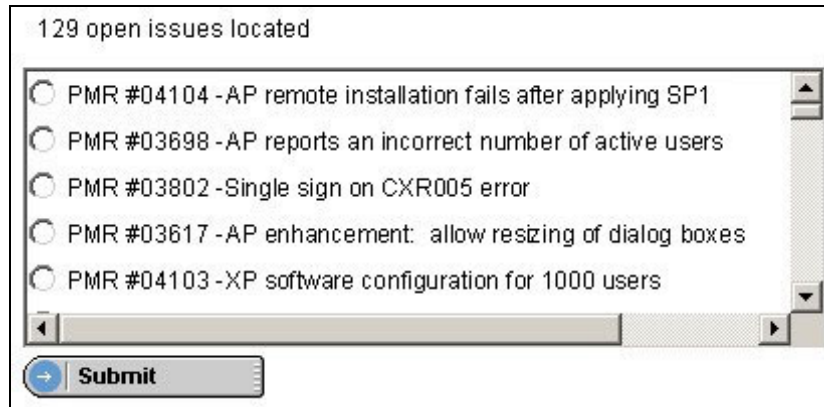


Figure B-3 Customer problem report selection screen

This screen makes use of standard AWT components.

The customer selects the problem report of interest and hits the **Submit** button. A new window appears and the applet waits for an agent to respond. Figure B-4 shows a chat dialog box after an agent responds.

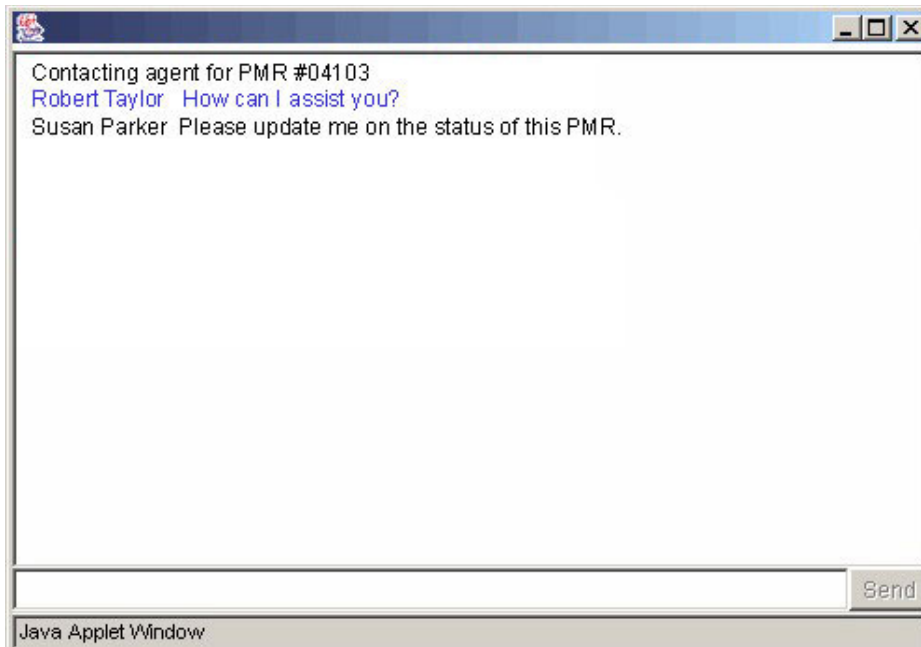


Figure B-4 Chat dialog of agent responding to a client

Within the chat dialog screen, we use the Sametime UI class ChatArea to display the chat transcript and standard AWT components for the input field and button. The **Send** button sends a line of text to the agent.

Customer attributes

The customer applet sets the attributes shown in Table B-1 so they can be displayed in the agent applet.

Table B-1 Attributes for the customer applet

Attribute constant	ID	Type	Description
STATE	1	INT	Applet state (1=ready, 2=requesting chat, and 3=chatting)
NAME	2	STRING	Customer name
COMPANY	3	STRING	Company name
INCIDENT_ID	4	STRING	Selected problem report number
PROBLEM_AREAS	5	STRING	General category of the selected report
DESCRIPTION	6	STRING	One line description of the selected problem
AGENT	7	STRING	Agent assigned to the selected problem report

Customer state machine

The customer applet has three states, as shown in Figure B-5 on page 461.

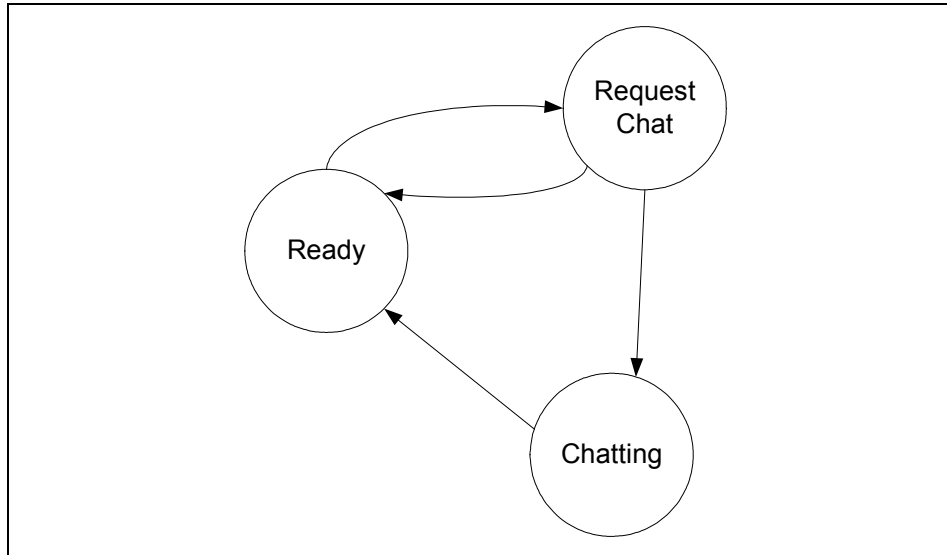


Figure B-5 Customer states

Where:

- ▶ Ready: The customer applet has loaded and the customer has not yet requested assistance. The agent applet ignores customers in this state (that is, they do not appear in the monitoring screen).
- ▶ Request Chat: The customer is requesting assistance. Agent applets detect this state change and shows the customer in the monitoring view. A customer can transition back to the ready state by closing the chat window. When an agent responds to a chat request, the customer applet goes into the chatting state. The customer applet only accepts a chat in this state, and accepts only one chat.
- ▶ Chatting: The customer is currently in a chat session with an agent. Once the chat finishes, the customer transitions back to the ready state.

The agent applet

The following sections describe the components that make up the agent applet.

Agent UI

When an agent logs onto the system the applet presents the monitoring screen (Figure B-6 on page 462). In one pane, this screen shows all agents on the

system and whom they are in contact with. The agent applet obtains this information by monitoring the users in the stage section and listening for their attribute changes. The other pane shows the customers currently requesting assistance. The applet obtains the customers by monitoring the customer section.

The Agent UI Applet interface is divided into two main sections. The top section, under the 'Monitoring' tab, displays a table with columns: Agent, Status, Customer, and Company. The bottom section, under the 'Chat' tab, displays a table with columns: Customer Name, Time in Queue, Company Name, Problem Area, Analyst, and Incident Number. An 'Accept Chat' button is located in the bottom right corner of the chat section.

Agent	Status	Customer	Company
Robert Taylor	Ready		

Customer Name	Time in Queue	Company Name	Problem Area	Analyst	Incident Number
Susan Parker	0:58	Joe's Software Emporium	Infrastructure	R. Taylor	04103

Accept Chat

Monitoring Chat

Figure B-6 Agent UI Applet

The agent selects one of the available customers and hits the “Accept Chat” button. Once the system establishes a chat, the applet switches over to the chat tab, as shown in Figure B-7.

The Chat area for agent interface shows a chat history window with two messages: 'Robert Taylor: How can I assist you?' and 'Susan Parker: Please update me on the status of this PMR'. Below the chat history is a text input area labeled 'Type your text'. To the right of the input area are buttons for 'Send', 'Send Page', 'Hang Up', 'Select All', and 'Copy'. The 'Chat' tab is selected at the bottom.

Robert Taylor: How can I assist you?
 Susan Parker: Please update me on the status of this PMR

Type your text

Send Send Page Hang Up

Select All Copy

Monitoring Chat

Figure B-7 Chat area for agent

Similar to the customer chat window, we use the ChatArea class to display the chat transcript and standard AWT components for the input field and buttons. The Send button performs the same function, sending a line of text to the customer. The Send Page button brings up a dialog box with a predefined list of URLs to send the customer. The Hang Up button terminates the chat, and the Select All and Copy buttons allow the agent to easily copy the contents of the chat transcript to the clipboard.

Agent attributes

The agent applet sets the attributes shown in Table B-2.

Table B-2 Agent applet attributes

Attribute constant	ID	Type	Description
STATE	1	INT	Applet state (1=ready, 2=responding to chat, and 3=chatting)
NAME	2	STRING	Agent name
CUSTOMER_NAME	3	STRING	Name of the current customer
COMPANY	4	STRING	Company name of the current customer

Agent state machine

The agent applet has three states, as shown in Figure B-8 on page 464.

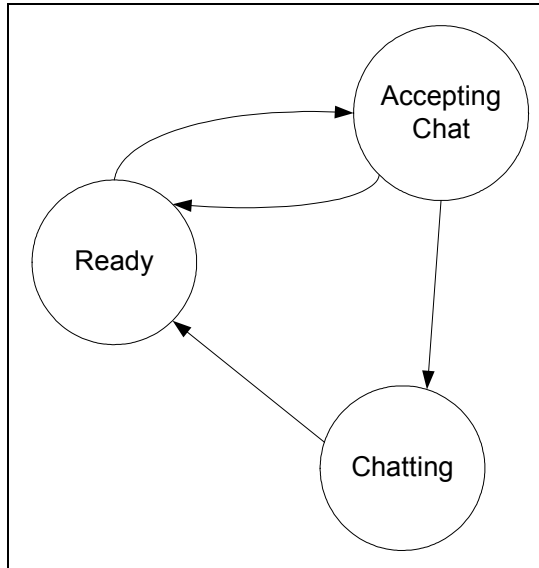


Figure B-8 Agent States

Where:

- ▶ Ready: The agent applet is ready to respond to chat requests.
- ▶ Accepting Chat: The agent is in the process of accepting a chat (see “Customer/Agent handshake” on page 464). From this state, the agent transitions to chatting if the customer accepts the chat or back to ready if the customer applet rejects the chat.
- ▶ Chatting: The agent is currently chatting with a customer. Once the chat finishes, the customer transitions back to the ready state. In this state, the agent sets the CUSTOMER_NAME and COMPANY attributes so this information displays on the other agent monitoring screens.

Customer/Agent handshake

The customer appears on all the agent screens. There is a possibility that more than one agent could choose to respond. We must ensure that only one establishes a link. We do this through a handshake between the customer and agent. The customer applet accepts one and only one connection. This is illustrated in Figure B-9 on page 465.

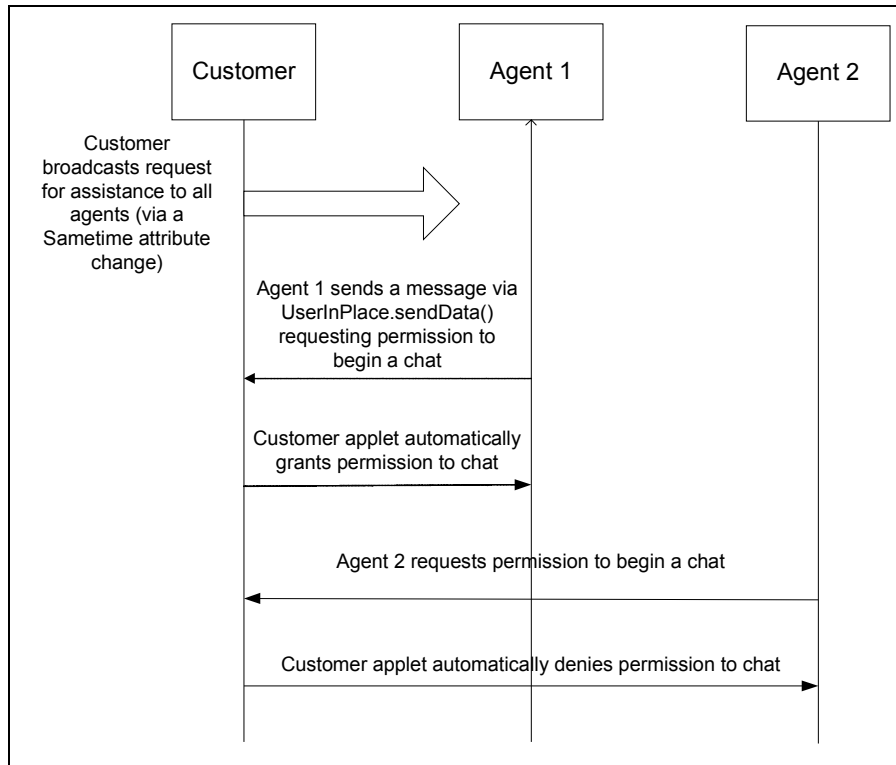


Figure B-9 Handshake between customer and agent

Helper classes

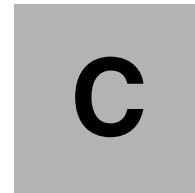
We developed the following wrapper classes for this application. In addition to their primary function, as described below, each class checks the return values from Sametime and report any errors.

Back-end classes

- ▶ **STcSession:** A small wrapper class for the Sametime `STSession` class. The main purpose of this class is to load all the components common to both applets (`STBase` and `Places`), hold references to the common services (`CommunityService` and `PlacesService`), and provide a simple interface to create `Places`.
- ▶ **STcPlace:** This much larger wrapper class for the Sametime `Place` object keeps track of all `Places` created, traps the `sectionAdded` events, and

identifies the sections. See Chapter 8, “Places and Place awareness” on page 183 for the relevant code.

- ▶ STcPlaceListener (interface): The STcPlace class has its own listeners, which implement this interface. With its own listeners, we can completely hide the identification of sections from caller classes.
- ▶ SectionMonitor
- ▶ STcApplet
- ▶ STcUser: A wrapper for the Sametime UserInPlace class.
- ▶ Agent
- ▶ Customer



Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG247037>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG247037.

Using the Web material

The additional Web material that accompanies this redbook includes the files shown in Table C-1.

Table C-1 Listing of sample code files

File name	Description
ch03_bot.zip	<p>This ZIP file contains subdirectories for the three bot examples in this chapter:</p> <ul style="list-style-type: none">▶ EchoBot: Contains EchoBot.java.▶ FAQBot: Contains SametimeBot.java and FAQBot.java. It also contains FAQBotConfig.nsf, the Domino database used by the bot.▶ TranslatorBot: Contains SametimeBot.java and TranslatorBot.java.
ch04_webservices.zip	<p>This ZIP file contains the UserStatus.java file that is turned into a Web service.</p>
ch06_workflow.zip	<p>This ZIP file contains the following files:</p> <ul style="list-style-type: none">▶ RBWorkflow.nsf: This is the Domino database referenced in Chapter 6, “Sametime and workflow” on page 139. It contains the AnnouncementSender and AnnouncementWebService Java agents.▶ AnnouncementSender.java: This is the Java application that is used to create the AnnouncementSender Web service.▶ ExcelAnnouncementExample.xls: This is the Microsoft Excel spreadsheet containing VBA code to access the AnnouncementSender Web service.
places_awareness.zip	<p>This zip file contains the code and class files to install both the server and client components of the Places sample discussed in Chapter 8, “Places and Place awareness” on page 183.</p>

File name	Description
st_enabling_portlets_samples.zip	This ZIP file contains the Sametime-enabled name list portlets (.war files) used in Chapter 10, “Sametime-enabling portlets” on page 273. It contains both versions of the name list portlet: one which has not yet been Sametime-enabled, and the version that has been Sametime-enabled.
meeting_center_customizations.zip	This directory contains code and examples used in Chapter 11, “Customizing the Online Meeting Center” on page 297.
ch12_customization_integration.zip	This ZIP file contains four directories: <ul style="list-style-type: none"> ▶ RichTextBot: Contains the RichTextBot.java file for the bot. ▶ RichTextClient: Contains the RichTextClient.java and ImageCanvas.java files that make up the applet. Also contains the various images used by the sample. ▶ TokenGenerator: Contains the TokenGenerator.java file for the TokenGenerator servlet. ▶ ASP: Contains the STLinksinASP.asp file and the TokenGeneratorServletforASP.java file.
non-st-jars.zip	Contains the non-stand jars (open source files) used in specific examples.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 473. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *B2B Collaborative Commerce with Sametime, QuickPlace, and WebSphere Commerce Suite*, SG24-6218
- ▶ *Domino Web Service Application Development for the IBM @server iSeries Server*, SG24-6862.
- ▶ *How to Portalize Your Existing Web Applications*, TIPS0028
- ▶ *IBM WebSphere Portal V4 Developer's Handbook*, SG24-6897
- ▶ *IBM WebSphere Portal V4.1 Handbook Volume 1*, SG24-6883
- ▶ *IBM WebSphere Portal V4.1 Handbook Volume 2*, SG24-6920
- ▶ *IBM WebSphere Portal V4.1 Handbook Volume 3*, SG24-6921
- ▶ *Lotus Sametime 2.0 Deployment Guide*, SG24-6206
- ▶ *Portalizing Domino Applications for WebSphere Portal*, SG24-7004
- ▶ *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292
- ▶ *WebSphere Portal 4.12 Collaboration Services*, REDP0319
- ▶ *WebSphere Version 5 Web Services Handbook*, SG24-6891
- ▶ *Working with the Sametime Client Toolkits*, SG24-6666
- ▶ *Working with the Sametime Community Server Toolkit*, SG24-6667

Other publications

These publications are also relevant as further information sources:

- ▶ *Sametime 3.1 Release Notes*, found at:
<http://www-12.lotus.com/ldd/doc/sametime/3.1/strn31.nsf/NNew?OpenNavigator>

The following publications are included with the Sametime Software Development Kit (SDK), which is installed on your Sametime server. You can find out more about accessing these documents in 2.2.1, “Sametime Software Development Kit (SDK) documentation” on page 22.

- ▶ *Sametime 3.0 Directory and Database Access Toolkit Developer's Guide*
- ▶ *Sametime 3.1 Administration Guide*
- ▶ *Sametime 3.1 C++ Developer's Guide*
- ▶ *Sametime 3.1 Installation Guide*
- ▶ *Sametime 3.1 Java Toolkit Developer's Guide*
- ▶ *Sametime Links 3.1 Toolkit Developer's Guide and Reference*
- ▶ *Sametime Links JavaScript API Reference*

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ alphaWorks NotesBuddy
<http://www.alphaworks.ibm.com/tech/notesbuddy>
- ▶ Apache Web Services - SOAP
<http://ws.apache.org/soap/index.html>
- ▶ IBM developerWorks: WebSphere Portal Zone
<http://www7b.software.ibm.com/wsdd/zones/portal>
- ▶ IBM - LDD Today: Adding a pop-up menu to your Sametime links
http://www.lotus.com/ldd/today.nsf/lookup/ST_popup
- ▶ IBM - Lotus Sametime support
<http://www-3.ibm.com/software/lotus/support/sametime/support.html>
- ▶ IBM Redbooks Site - Redbooks, Redpapers, Hints and Tips
<http://www.redbooks.ibm.com>

- ▶ IBM - Sametime Fix List Database
<http://www-10.lotus.com/ldd/stfixlist.nsf>
- ▶ IBM - WebSphere Portal for Multiplatforms
<http://www.ibm.com/software/websphere/portal>
- ▶ InfoCenter for IBM WebSphere Portal for Multiplatforms Version 4.2.1, Experience offering
<http://publib.boulder.ibm.com/pvc/wp/42/exp/en/InfoCenter/index.html>
- ▶ InfoCenter for IBM WebSphere Portal for Multiplatforms Version 4.2.1, Extend offering
<http://publib.boulder.ibm.com/pvc/wp/42/ext/en/InfoCenter/index.html>
- ▶ Lotus Developer Domain, Sametime Developers' Forum
<http://www.lotus.com/ldd>
- ▶ Lotus Sametime Product Page
<http://www.lotus.com/sametime>
- ▶ Technote # 1101472
<http://www-1.ibm.com/support/docview.wss?uid=swg21101472>
- ▶ WebSphere Application Server Infocenter
<http://www-3.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ WebSphere Translation Server
http://www-3.ibm.com/software/pervasive/ws_translation_server/
- ▶ XMethods home page
<http://www.xmethods.net>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads:

ibm.com/support

IBM Global Services:

ibm.com/services

Index

A

- active call center 408
- Active Server Pages (ASP) 351
- activities 183
- ActivityHandler class 209
- ActivityHandler object 209
- ActivityServiceListener 204
- addImServiceListener() method 55
- adding menu options to Sametime Links 259
- Administration 393
- AIX 412
- announceFrame.html 229
- Announcement Service 141
- AnnouncementSender Web service 148, 150
- Anonymous logins 414
- APIs
 - new features in Sametime 3.1 APIs 12
- attributes within Sametime Virtual Places 183
- Audio 371
- awareness 140, 219
- AwarenessList class 173, 176

B

- B2B 365
- B2C 365
- BabelFish Web service 74
- background-color 376
- BLServiceListener class 173
- Borland JBuilder 31
- bots 49, 394
 - common framework 52
 - creating a Sametime session 52
 - definition of a bot 50
 - developing 51
 - Directory Bot 60
 - Echo Bot 256
 - enhancing the bot framework 79
 - examples 56
 - Echo Bot 56
 - FAQ Bot 61
 - Translator Bot 56, 74
 - listening for incoming messages 55
 - logging in to the community 53

- overview of bots 50
- presence 50
- Registering the message type 54
- Responding with logic 55
- RichTextBot 334
- toolkits for building bots 51
- using STLinks and bots 256
- Branding the Sametime Meeting Center 297–298
 - changing the look 302
 - Java Server Page (JSP) front-end 305
 - Meeting Summary Email 305
 - typical reasons for branding 298
- buddylist 384
 - loading 175
- BuddyList service 163
 - advantages over storage service 164
 - BL structure 165, 177
 - handling of public groups 165
 - handling private groups 165
- Business context
 - phased approach building solution 368
- Business context scenario
 - ability to chat for online support 400
 - customizing the meeting center look and feel 373
 - Enabling the Customer Tracking application 381
 - future planned enhancements 422
 - implementing offline message delivery 384
 - implementing presence awareness 380
 - Implementing the offline messenger bot 387
 - integrating Sametime into workflow 391
 - logging of online customer conversations 411
 - providing users with self service tools 393

C

- C++ 408
- C++ Toolkit
 - Basic Buddy List sample 165
 - Extended Live Names sample 163
- chat logging 127, 411
 - appropriate use review 130
 - asynchronous implementations 131

- collaborative commerce rules and workflow 129
- corporate and public sector governance 129
- customizing 136
- developer considerations 130
- modes 131
- regulatory compliance 128
- synchronous implementations 131
- toolkit examples 132
- chat logging SPI (Service Provider Interface) 127
- Collaboration
 - benefits and importance 5
- Collaboration Center 279
 - My Lotus Team Workspaces 280
 - People Finder portlet 280
 - Web Conferencing portlet 280
- COM 408
- COM Toolkit
 - Installing 42
- Community Server Toolkit 414
- Community Server Toolkit Tutorial 385
- Community services 6
- CommunityService 88
- CommunityService object 53
- CommunityTrustedIP field 191
- createPersistentPlace() method 207
- Creating a Sametime session 52
- CSEnvironment 283
- CSS 376
- Customization and Integration services 6, 9
- customizing the Sametime Meeting Center 297, 373
 - changing the look 302
 - Meeting Summary Email 305
- customizing the sametimelogo.gif 376

D

- Database and Directory Assistance Toolkit (DDA) 127
- DB/2 396
- DDA Toolkit 127
- DebugLevel.class file 226
- Deployment 393, 471
- development environment
 - setting up 22
- DLL 412
- Domino java agent 142
- Domino workflow application 139

E

- educational sessions 413
- enabling live names via Sametime Links 219
- enabling with Sametime Links 351

F

- FAQ 396

G

- getUserStatus() method 88
- gif images
 - customizing the meeting center look and feel 376

H

- hostInfo.js 226
- HTML HEAD 381

I

- IBM AIX 369
- IBM Lotus Instant Messaging and Web Conferencing
 - overview 3
 - See also Sametime
- IBM OS/390 369
- IBM OS/400 369
- IBM WebSphere Portal V 4.1 Handbook Volume 1 274
- IBM WebSphere Portal V4 Developer's Handbook 274
- ImListener interface 55
- ImReceived 388
- imReceived() method 55
- ImServiceListener interface 55
- Informix 396
- Input.jsp 110
- Installation 393
- instant collaboration 273
- Integrating with Microsoft Excel 155
- Intel 369
- isLoggedIn() method 90

J

- Java 408
- Java Client Toolkit 141
- Java interfaces 396

JBuilder 31
JDK 35

L

language 417
Lightweight Third Party Authentication 342
loadSemanticComponents() method 53
loggedIn() event 143
Logging in to the community 91
Logging into the community 53
login() method 90
LoginListener interface 87
logon 375
LookupService 88
Lotus Domino 368
Lotus Instant Messaging 368
Lotus Web Conferencing 368
LTPA 342
LTPA tokens
 using for Single Sign-On (SSO) 342

M

Menu tags 285
Message types
 IM_TYPE_CHAT 54
Method.jsp 110
Microsoft Visual Basic 43
Microsoft Visual C++ 39
Microsoft XML services 156, 158
Moderated Chat example
 see also Places and Place architecture
MS SQL 396

O

offline message delivery 383
Online Meeting services 6
openPlaceWin() 406
Operating System 368
Oracle 396

P

PeopleService tags 284
PeopleSoft 396
Place awareness 183
Place member attributes 457
Place type
 value assigned 195

PlaceMonitor class 209
Places 183
 activities 183, 189
 attributes 183, 189
 example
 audience section 189
 building the server-side 203
 lobby 188
 managing the queue 212
 queue 188
 setting environment variables 190
 stage 188
 key concepts 184
 monitoring the Place 209
 persistent Place 203, 206
 Place characteristics 206
 scenario example 185
 sections 183, 187
 getting references to 196
Places architecture 183
PlacesAdminService components 203
PlacesAdminServiceListener 204
Portal
 Collaborative Components API 273, 282
portal,Development Community 424
portalizing existing Web applications 424
portalizing,Applications for WebSphere Portal 471
portalizing,Existing Web Applications 471
portlet,cs.jar 287
portlet,JSP 286
portlet,people.tld 287
portlets 273
Presence/Awareness 5

Q

QueueMonitor class 211

R

Realtime collaboration
 See Community services
Red Hat Linux 369
redbook
 Portalizing Domino Applications for WebSphere
 Portal, SG24-7004 424
Redbooks Web site 473
 Contact us xvi
resolve() method 143
ResolveListener interface 87, 93

Result.jsp 110
Reverse proxy support 13
RichTextClient Sametime applet 337

S

Sametime

- customization and integration ideas 333
- enabling functionality within a portlet
 - Collaborative Components approach 281
 - Sametime Links approach 292
- overview 3
- workflow applications 140

Sametime 3.1 APIs 12

Sametime and workflow 139

Sametime Bots 49

- see bots

Sametime C++ Toolkit 10

Sametime COM Toolkit 10

Sametime Community Server 51

Sametime Community Server Toolkit 11

Sametime Connect client 333

Sametime deployment 367

Sametime Directory and Data Access Toolkit 11

Sametime functionality

- embedding functionality into a Notes workflow application 148

Sametime Java Toolkit 11

Sametime Links 139, 153, 219, 410

- Adding menu options 259
- announceBtn.html 228
- anonymous users 222
- borderFrame.html 226
- branding the ST Links experience 245
- building an interactive Web site 244
- changeStatus.html 230
- chatApplet.html 231
- chatBtn.html 231
- chatWindow.html 231
 - changing 257
- deployment considerations 220
- dirApplet.html 232
- enabling live names 219, 223
- HTML files 226
- im.html 233
- inputFrame.html 234
- invitation.html 235
- invitees.html 236
- inviteFrame.html 236

- inviteOthers.html 237
- nway.html 239
- offlineLink option 262
- overview 220
- peopleHeader.html 240
- peoplelist.html 241
- place.html 241
- placeChat.html 242
- placeHeader.html 242
- platform support 221
- removing the buttons 246
- res.js 243
- resolve.html 243
- scalability upgrade in 3.1 15
- see also STLinks
- statusFrame.html 227
- Taking it further 255
- transcript.html 227
- using with bots 256

Sametime Links Toolkit 9, 219

- Multiplatform support 14

Sametime listener interfaces

- LoginListener 87
- ResolveListener 87
- StatusListener 87

Sametime services 6

- Community services 6
- Customization and Integration services 6
- Online Meeting services 6

Sametime session

- creating 52, 88

Sametime Web service 85

Sametime-specific packages 141

SAP 396

SDK

- Listing of toolkits installed on Sametime Server 22, 386

Sections 183

Sections within Places 183

sendAnnouncement() method 142

ServerAppService component 203

Siebel 396

Simple Object Access Protocol

- see SOAP

Single Sign-On (SSO)

- alternative approaches 341

SOAP 84–85, 106, 150, 160

Social network analysis 430

source code 286

- SPI 411
- StatusListener interface 87, 94
- stbackground.gif 376
- StChatLogDummy.dll 412
- StChatLogFile.dll 412
- STComm.jar 221
 - including in Notes Class Path 144
- stconfig.nsf 191
- STExtendedAttributes 457
- STLinkClicked event 263
- STLinks 153
- STLinks directory
 - contents 226
- stlinks.cab 226
- stlinks.css 226
- stlinks.jar 226
- Stlinks.js 226
- STLinksEnterPlace() 403
- STLinksetMyStatus() 410
- STLogonForm 376
- storage service 163
- STRes.jar files 221
- STSession class 53
- STSession object 53
- STSession.getCompApi() method 53
- STUser object 93
- Sun JDK 35
- Sun Solaris 369
- supported operating systems 368
- SuSE Linux 369

T

- tag language descriptors (TLDs) 284
- TestClient.jsp 110, 149
- textReceived 390
- TokenGenerator servlet 342
- Toolkits
 - Installing 22
 - installing Microsoft Visual C++ 39
 - Installing the C++ Toolkits 37
 - Installing the COM Toolkit 42
 - Overview of the Sametime 3.1 Toolkits 9
 - Sametime C++ Toolkit 10
 - Sametime COM Toolkit 10
 - Sametime Community Server Toolkit 11
 - Sametime Directory and Data Access Toolkit 11, 127
 - Sametime Java Toolkit 11

- Sametime Links Toolkit 9
- Town Hall 414
- Translation Server,Java code 421

U

- UDDI 85
- UN Discussion
 - Places example 415
- Universal Description, Discovery, and Integration (UDDI) 85
- User's status
 - object representing status 88
- UserStatus application 86
- UserStatus class 87

V

- video 371
- Visioning Scenario 359
- Visual Basic 43
- Visualizing Sametime
 - application overview 429
 - architecture 430
- VPS_Trusted_IPS 191

W

- W32 dll for chat logging 412
- WatchList object 93
- Web conferencing functionality 5
- Web services 392
 - BabelFish Web service 74
 - creating the user status 97
 - deploying the UserStatus Web service 114
 - exposing existing Sametime applications as Web services 97
 - fundamentals 84
 - jar files 152
 - overview 83
 - testing Sametime Web service 122
 - usage with Sametime 83, 85
- Web Services Description Language 85
- WebSphere 273
- WebSphere Application Server 114
- WebSphere Portal 273, 368
 - Collaborative Components API 282
- WebSphere Portal Info Center 274
- WebSphere Studio Application Developer 5.0 26, 83, 86

WebSphere Translation Server 418
WebSphere,Portal 273
WebSphereTranslation Server 418
Windows 2000 369
Windows NT 369
workflow 139, 275, 390
 AnnouncementSender application 141
 offline message delivery 383
 sending an tag 154
 sending an HTML tag 154
writePlaceCounter() 406
writeSametimeLink() 224, 382
writeSTLinksApplet() 404
WSDL 85

X

XML 84–85
XML schema 106

Z

zSeries 369



Redbooks

Lotus Instant Messaging/Web Conferencing (Sametime): Building Sametime-Enabled Applications

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



Redbooks

Lotus Instant Messaging/ Web Conferencing (Sametime): Building Sametime-Enabled Applications

Build custom bots

Advanced capabilities through Sametime Links

Integrate Sametime into workflow applications

This IBM Redbook builds upon two existing Redbooks: *Working with Sametime Client Toolkits*, SG24-6666 and *Working with the Sametime Community Server Toolkit*, SG24-6667. Focusing on the latest toolkits available for IBM Lotus Instant Messaging and Lotus Web Conferencing (Sametime) 3, it explores Sametime as a development platform, illustrating how to extend the functionality of Sametime beyond its more commonly known instant messaging and e-meeting hosting features.

It provides a detailed technical discussion and examples for building and integrating Sametime real-time collaborative capabilities, presence awareness, and Web conferencing capabilities into both new and existing applications. In-depth discussions and code examples are provided for topics such as building custom bots, leveraging Sametime functionality through Web Services, and integrating Sametime into workflow applications. Additionally, this redbook reveals how you can customize (brand) your organization's Sametime Meeting Center to more closely match the company's identity. Finally, thorough analysis is given to the topics of Sametime Links and the Sametime Places architecture. Several samples of Sametime applications, with source code, are included.

Ultimately, we hope this redbook will help you better appreciate how your organization may benefit by more effectively leveraging Sametime.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks